To the Graduate Council:

I am submitting herewith a thesis written by Stephen W. Kercel entitled "A Sensor Using Raman Spectroscopy for On-Line Composition Monitoring of Distillation Mixtures in Real Time." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Michael J. Roberts, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Stephen W Keen

Date August 1990

A SENSOR USING RAMAN SPECTROSCOPY FOR ON-LINE COMPOSITION MONITORING OF DISTILLATION MIXTURES IN REAL TIME

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Stephen W. Kercel

August 1990

DEDICATION

This work is dedicated to our fellow predator, the grizzly bear (ursus arctos), a creature who occupies the same ecological niche as ourselves. If we so change the world that no place remains for the great bear, will we ourselves survive for long?

ACKNOWLEDGMENTS

I would like to thank Dr. Michael J. Roberts, who had the dual role of Major Professor for my graduate work and Principal Investigator for the On-Line Chemical Composition Analyzer Development Project, for his guidance, criticism, and invaluable assistance in this research.

For their guidance in the preparation of this thesis, I thank the other members of my Graduate Committee, Dr.Igor Alexeff, Dr. Theron Blalock, and Dr. David Straight.

The work recounted in this thesis would not have been possible outside the context of the On-Line Chemical Composition Analyzer Development Project. I am indebted to the other team members, Dr. Arlene Garrison, Dr. Dave Trimble, Dr. E. C. "Bud" Muly, Dr. Charlie Moore, and Mr. Miroslaw Bialkowski.

I thank Alan Ganz and Bob Bennett of Perkin Elmer for the technical details on the laboratory apparatus.

The pursuit of an advanced degree is always a major undertaking, but never more so than when the pursuit begins at midlife. Without the support and encouragement of my wife, Nancy Lauckner, this effort would not even have begun.

ABSTRACT

The goal of this project is energy conservation. All the discoveries made during the course of the research are incidental to this goal. Distillation is a notable example of a commercial process that can be dramatically improved through energy conservation. To efficiently operate a distillation column requires the ability to accurately monitor the composition of the distillation mixture. It has been found that Raman spectroscopy, a proven laboratory technique for composition analysis, can be applied to industrial composition monitoring.

Practical implementation of Raman spectroscopy in an industrial setting is constrained by several competing limitations. The instrumentation must be housed in a benign environment, while the process, a hostile environment, must be monitored on-line with a hardened probe. This requires that the instrumentation and the probe be connected by optical fibers. Raman scattering is a weak effect, but only a small fraction of the available Raman energy can be coupled into the fibers. Consequently, in order to obtain usable spectra, stringent demands are imposed on the optical efficiency of the system. Finally, the instrumentation is expensive, but the market will not bear a high price for composition measurements. An acceptable cost per monitoring

point can be realized by using time division multiplexing to allow the same instrumentation to monitor several points simultaneously.

All the elements of a prototype composition monitoring system have been constructed and tested in the laboratory. The composition readings produced by the system are of sufficient accuracy to be used for efficient control of a distillation column. The computerized control system has been verified with the hardware. The system is ready to be applied to an actual distillation column.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	1
2 EXPERIMENTAL DEMONSTRATION OF FEASIBILITY	11
3 DISCOVERIES MADE DURING SHORT FIBER EXPERIMENTS	29
4 PROPERTIES OF LONG FIBERS	53
5 COUPLING A LASER TO A FIBER	72
6 THE FIBER OPTIC PROBE AND SAMPLE CELL	81
7 COUPLING A FIBER BUNDLE TO AN INSTRUMENT	101
8 CONTROL SOFTWARE	117
9 HARDWARE INTERCONNECTIONS	144
10 CONCLUSIONS	149
LIST OF REFERENCES	153
APPENDICES	162
HARDWARE SPECIFICATION	163
CONTROL SYSTEM SOFTWARE SPECIFICATION	168
EXPERIMENTAL DATA: REPRESENTATIVE SPECTRA	181
EXPERIMENTAL DATA: CHEMOMETRIC ANALYSIS	199
EXPERIMENTAL DATA: CENTER OF SCATTERING	225
EXPERIMENTAL DATA: NUMERICAL APERTURE	238
EXPERIMENTAL DATA: CONNECTION LOSS MEASUREMENT	251
CODE LISTING: PROBE ANALYSIS PROGRAM	262
CODE LISTING: DOUBLE LENS COUPLER DESIGN PROGRAM	291
CODE LISTING: PROTOTYPE I/O PROGRAMS	326
CODE LISTING: INSTRUMENT CONTROL PROGRAM	340
VITA	429

LIST OF TABLES

3-1. Attenuation Through Quartz Window 30
3-2. Comparative Raman Peak Values 40
4-1. Intensities of Acetic Acid Spectra 55
4-2. Signal to Noise Ratios 57
4-3. Height Versus Excitation 65
4-4. Silica Raman Intensity 67
5-1. Reflection Limited Minimum Losses 76
5-2. Observed Coupling Efficiencies into Dow Probe . . 78
5-3. Fiber Output Radiation Patterns 79
6-1. Computed Coupling Coefficients 87
6-2. Comparative Predicted Probe Performances 91
6-3. O-Ring Compatibility with Sample 97
7-1. System Performance for Several Couplers 116
8-1. Control Computer I/O Map 119
10-1. Chemometric Errors Compared 152
D-1. Short Fiber Eastman Chemometric Errors 206
D-2. Long Fiber Eastman Chemometric Errors 210
D-3. Short Fiber Isopropanol Chemometric Errors 214
D-4. Short Fiber Hercules Chemometric Errors 218
D-5. Eastman Partial Least Squares Errors 222
E-1. Benzene Intensity Profile 231
F-1. Newport and Dow Fibers 242
F-2. Low Water Fibers 245
F-3. Numerical Apertures 249
G-1. Loss Statistics per Measurement 254
G-2. Cumulative Loss Statistics 254
G-3. First Connector Loss Measurement 257
G-4. Second Connector Loss Measurement 258
G-5. Third Connector Loss Measurement 259
G-6. Fourth Connector Loss Measurement 260

LIST OF FIGURES

1-1. Raman Spectrum of Benzene 4
1-2. On-Line Raman System 8
1-3. Azeotropic Distillation Column 9
2-1. Free Optics System 12
2-2. Short Fiber System 15
2-3. Fiber Optic Probes 16
2-4. Long Fiber Initial Setup 18
2-5. Accuracy of Chemometric Predictions 20
2-6. Cumene Oxidate and Acetophenone 26
2-7. Effect of Resolution 27
3-1. Response of Interferometer to White Light 33
3-2. White Light Response of Dow Probe 34
3-3. Wavelength Response of Dow Probe 35
3-4. Baseline for EIC Response 37
3-5. White Light Response of EIC Probe 38
3-6. Wavelength Response of EIC Probe 39
3-7. Non-Linearity of Isopropanol 42
3-8. Detector Noise 46
3-9. Raman Spectrum of Water 49
3-10. Raman Spectrum of Empty Quartz Cuvette 50
3-11. Power Drift of Benzene 51
4-1. Response of 35 m Fiber at 23°C 60
4-2. Response of 35 m Fiber at 40°C 61
4-3. Cyclohexane through Fiber at 23°C 62
4-4. Linearity of Raman Spectra 66
4-5. Silica Raman Line Height 68
6-1. Probe Coupling 82
6-2. Predicted Performance of Four Designs 92
6-3. Sample Head 95
6-4. Sealing Details 96
6-5. Sample Cell 99
7-1. Area and Angle of Acceptance 102
7-2. Fiber Bundle to Instrument Coupler 104
7-3. Cyclohexane Through Optimal Coupler 113
8-1. Data Flow Diagram of Control Software 128
8-2. Typical Cell Description File 132
8-3. Typical Spectrum Description File 136
B-1. Control System Interfaces 170
C-1. Acetic Extract - Free Optics 183
C-2. Acetic Extract - Dow Probe Short Fiber 184
C-3. Acetic Extract - Dow Probe Long Fiber 185
C-4. Acetic Extract - EIC Probe 186
C-5. Isopropanol - Free Optics 187
C-6. Isopropanol - Dow Probe Short Fiber 188
C-7. Isopropanol - Dow Probe Long Fiber 189
C-8. Isopropanol - EIC Probe 190
C-9. Cumene - Free Optics 191
C-10. Cumene - Dow Probe Short Fiber 192
C-11. Cumene - Dow Probe Long Fiber 193
C-12. Cumene - EIC Probe 194

C-13. Benzene - Free Optics 195
C-14. Benzene - Dow Probe Short Fiber 196
C-15. Benzene - Dow Probe Long Fiber 197
C-16. Benzene - EIC Probe 198
D-1. Cumene Sample Cell 201
D-2. Short Fiber Eastman RMS Error 207
D-3. Short Fiber Eastman Maximum Error 208
D-4. Short Fiber Eastman Std. Deviation 209
D-5. Long Fiber Eastman RMS Error 211
D-6. Long Fiber Eastman Maximum Error 212
D-7. Long Fiber Eastman Std. Deviation 213
D-8. Short Fiber Isopropanol RMS Error 215
D-9. Short Fiber Isopropanol Maximum Error 216
D-10. Short Fiber Isopropanol Std. Deviation 217
D-11. Short Fiber Hercules RMS Error 219
D-12. Short Fiber Hercules Maximum Error 220
D-13. Short Fiber Hercules Std. Deviation 221
D-14. RMS Error PLS vs. MLR 223
D-15. Maximum Error PLS vs. MLR 224
E-1. Benzene Profile Experiment 227
E-2. Representative Benzene Profile Spectra 229
E-3. Undesired Signals in Benzene Profile 232
E-4. Noise in Benzene Profile 233
E-5. Ratio of Desired to Undesired Signals 234
E-6. Benzene Intensity Versus Probe Depth 235
F-1. Numerical Aperture Measurement 240
F-2. Newport Fiber Numerical Aperture 243
F-3. Dow Probe Numerical Aperture 244
F-4. Low Water Numerical Apertures 248
G-1. Attenuation Statistics per Measurement 255
G-2. Cumulative Attenuation Statistics 256

Chapter 1

INTRODUCTION

Objective

The overall objective of this research is to adapt Raman spectroscopy, a proven laboratory technique, to industrial use. [Hecht 87, p. 554] Specifically, this means the construction of a system capable of sensing the composition of a liquid chemical mixture in a distillation column, and capable of generating composition data that can be used to control the distillation column. Constraints imposed by the industrial environment require that the system have shorter data collection times than the laboratory technique, operate on-line, and monitor several physically separate points simultaneously. The system is envisioned as being suitable for use with various distillation columns containing different liquid chemical mixtures. [Garrison 90]

Why Raman Spectroscopy?

Light scattering involves the absorption and nearly instantaneous reradiation of incident light. In a dense transparent medium, the dominant scattering mechanism is

Rayleigh scattering. This process involves the elastic scatter of incident light. In other words the absorbed radiation is reradiated at exactly the same energy or frequency. [Hecht 87, pp. 292-294]

Spontaneous Raman scattering is an inelastic mechanism. The radiation is scattered at slightly higher or lower energy than the incident light. A vibrating bond in a molecule (which can be viewed as electrons vibrating relative to stationary nuclei) can be treated as an oscillating electric dipole. This dipole can absorb energy from or donate energy to the incident photon. Therefore, when the photon is Raman scattered, the frequency of radiation is shifted by an amount equal to the energy of a molecular vibration. Because different molecules contain bonds that vibrate at characteristic frequencies, the Raman spectrum of a chemical is very characteristic of that compound. The Raman spectrum (the offset of the Raman scattered light frequency with respect to that of the absorbed light) of a material is also independent of the frequency of the incident light.
[Hecht 87, pp. 553-555]

Both Rayleigh and Raman scattering occur simultaneously. In both Rayleigh and Raman scattering the re-radiated photons propagate in random directions, and can be considered isotropic processes. Raman is the less efficient

phenomenon. Typically, for every million photons which are Rayleigh scattered, only one will be Raman scattered.

Raman scattering was chosen for this project because it allows for high resolution spectroscopy for a broad range of chemicals. The spectral features are reasonably intense and narrow as contrasted to a blank background. For example, Figure 1-1 shows a Raman spectrum for benzene. In a mixture, the spectral features of the different components may be very close together, but they can still be resolved since they are very narrow. Distinct features for a wide range of chemical samples appear within a limited bandwidth. [Garrison 89, pp. 45-59]

Other techniques were considered but rejected. [Garrison 89, pp. 5-15] Chromatographic methods are not suitable because data collection time can be lengthy and extensive plumbing is required to sample the process. Sampling is expensive, prone to failure, and effectively eliminates the analysis of liquids containing dissolved solids. Nuclear magnetic resonance uses intense electromagnetic fields that would be perturbed by stray RF and moving metal in an industrial environment. Mass spectrometers are suitable for industrial process control of gases, but not liquids. Absorption spectroscopy in

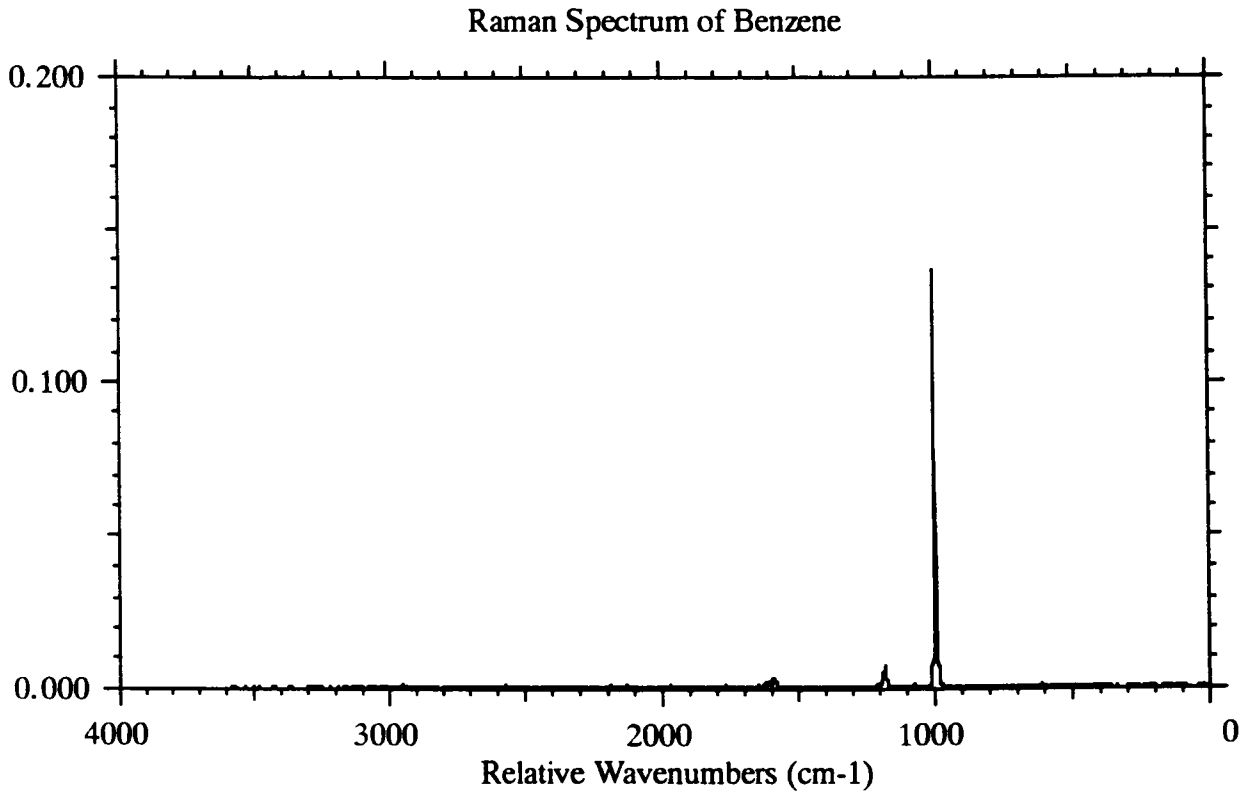


Figure 1-1. Raman Spectrum of Benzene

ultraviolet, visible or near infrared relies upon having an absorbing species present (often not the case) and lacks the resolution needed for distillation control. Mid-infrared absorption spectroscopy involves the use of brittle, water soluble window materials and expensive optical fibers. Reflection techniques will not work since the chemical samples of interest are clear.

Phases of the Project

The research is organized into three phases. [Garrison 89, pp. 81-82] Phase 1 investigated the practicality of Raman spectroscopy as a technique for sensing the composition of mixtures in a distillation column. This phase was started in March 1988 and was finished in July 1989. The work was conducted under DOE Contract DE-FC07-88ID12691.

The feasibility of using Raman spectroscopy to analyze distillation mixtures was demonstrated as follows: Several laboratory analyzers, including a table-top version of the on-line analyzer were used to analyze different concentrations of representative distillation mixtures. They produced chemometric composition predictions of sufficient accuracy to provide precise control of distillation columns. On this basis, Raman spectroscopy was judged to be a

feasible technique. The details of the phase 1 work are recounted in the literature and need not be repeated here. [Garrison 90]

Details of the development of the prototype system from the beginning of phase 2 through the testing of the system elements are covered in this thesis. The task in Phase 2 is to build, test, and refine a working prototype of a multiplexed on-line analyzer. This phase is currently under way, having started in August 1989. It will be completed in May 1991. This work is being conducted under DOE Contract DE-FC07-88ID12691 (extended).

The analyzer is based on Raman spectroscopy. It is capable of remotely monitoring the chemical composition at five different points, each located 10 meters or more from the spectroscopic equipment. After laboratory testing, it will be installed on the UT Chemical Engineering Department isopropanol column.

Phase 3 will consist of installing the prototype on industrial distillation columns and using the data generated by the Raman system for column control. It will start in June 1991. Funding of phase 3 depends on the successful completion of phase 2. No official end date for phase 3 has been established.

A schematic diagram, showing the system in an industrial setting, is shown in Figure 1-2. The system will monitor four columns and a calibration sample. Its first test in a commercial environment will be on a Tennessee Eastman column in Kingsport, Tennessee. A diagram of the Eastman column is shown in Figure 1-3.

Support

This research is being conducted by the UT Measurement and Control Engineering Center. It is primarily funded by the U.S.Department of Energy, Idaho Operations Office, to develop a new energy conservation technique. It is based on the observation that effective control of distillation columns would reduce the national energy consumption by 0.3% [DOE 87].

Facilities, chemical samples, and technical consultation are being provided by major chemical companies such as Tennessee Eastman and Hercules. In addition to energy cost reductions, they anticipate substantial cost savings from improved efficiencies in chemical production due to precise control. [Garrison 89, pp. 77-80] Equipment has been loaned and/or donated by manufacturers such as Dow Chemical and Perkin-Elmer.

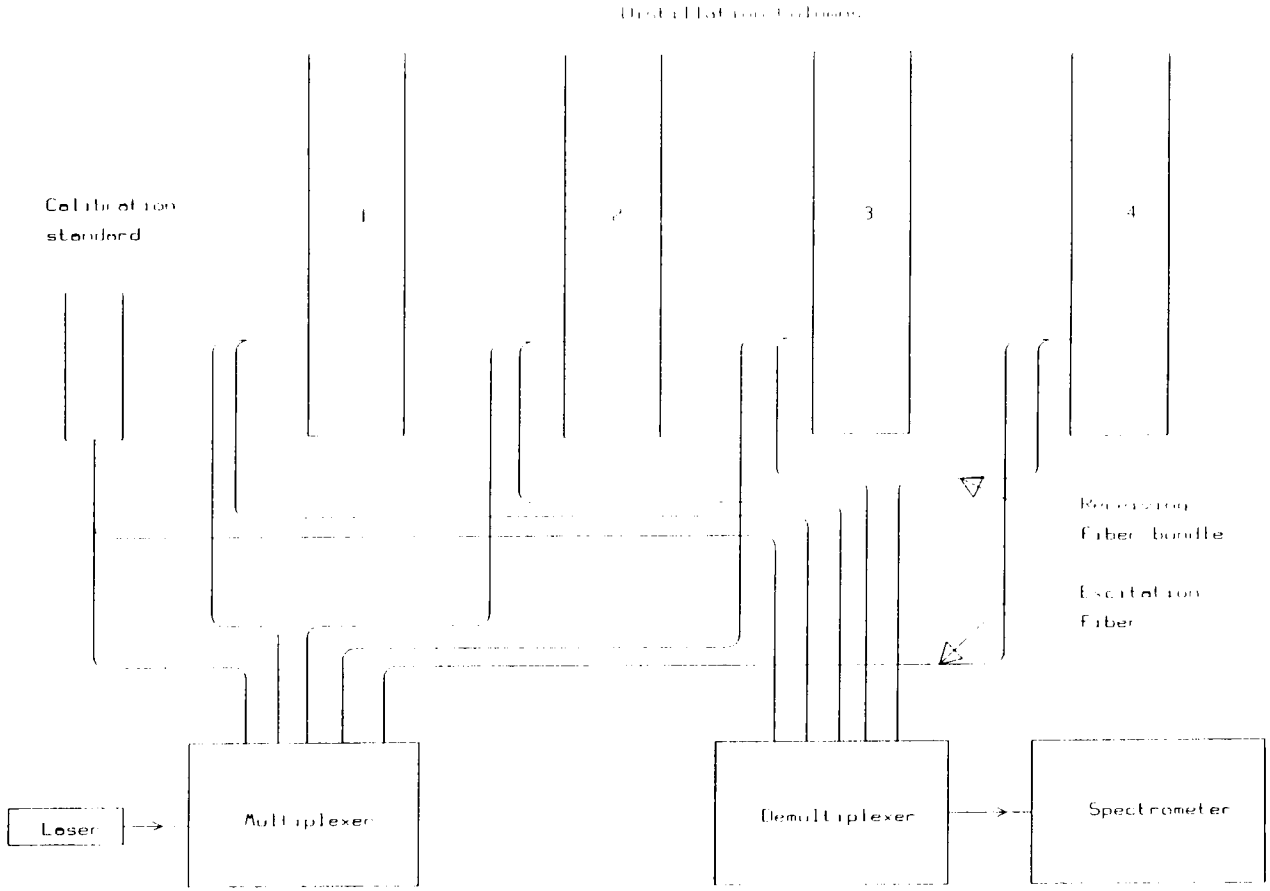


Figure 1-2. On-Line Raman System

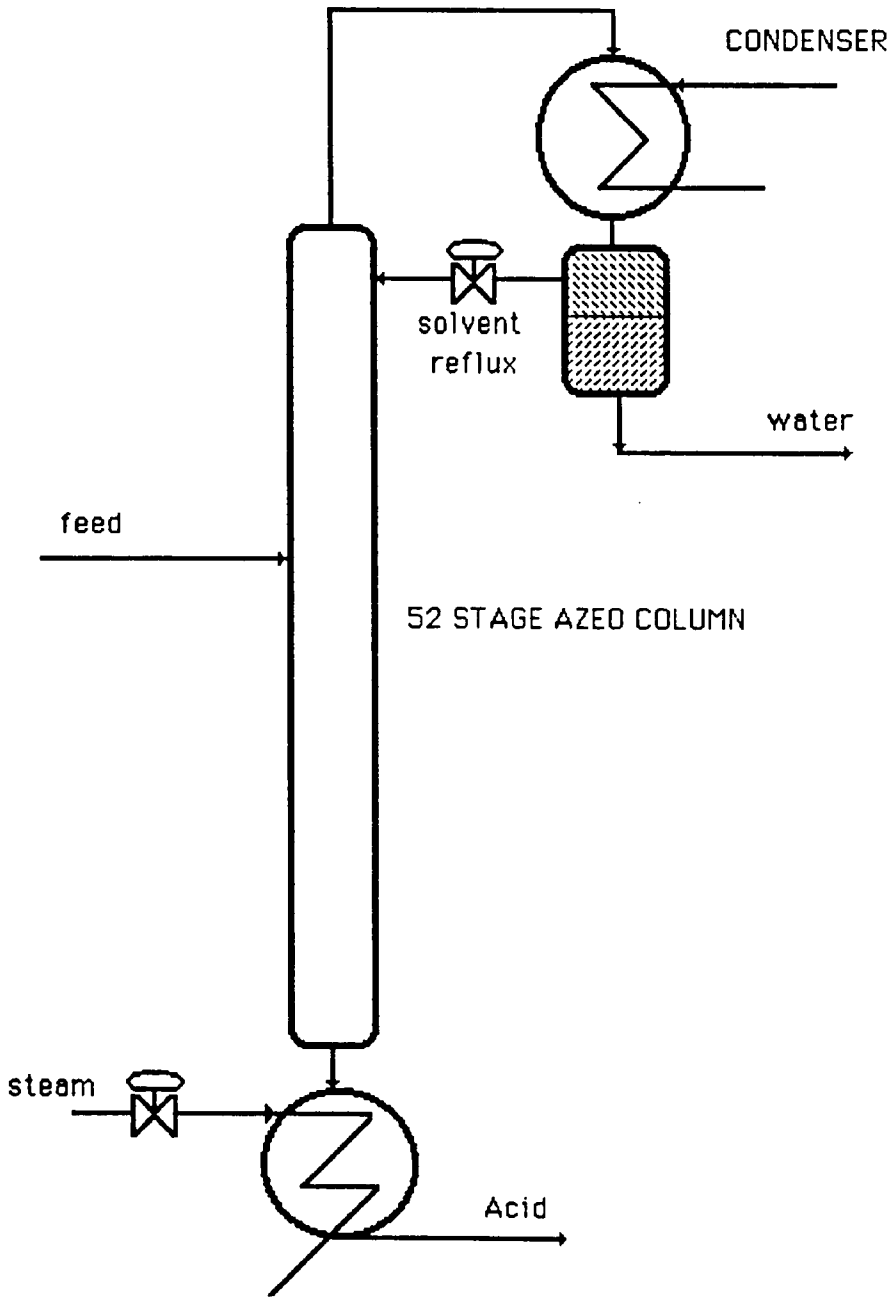


Figure 1-3. Azeotropic Distillation Column

Units of Measure

Several units of measure of the wavelength of light are used in this thesis. In describing the light output of a laser, the light is characterized by its free space wavelength, usually measured in microns (μ). In describing spectra, the units used are those commonly used by spectroscopists. Wavelengths in spectra are often measured in terms of wave numbers (wn or cm-1). Physically, the wave number of a given light beam is the number of waves of light per unit length. [Hecht 87, p. 16]

CHAPTER 2

EXPERIMENTAL DEMONSTRATION OF FEASIBILITY

What Demonstrates Feasibility?

A feasible composition monitor is one that produces chemometric predictions within three minutes with a total RMS error (not relative error) of two percent or less. [Garrison 89 p. 4, 75-78] Several different systems have been constructed and used to collect Raman spectra for typical distillation mixtures. These spectra have been analyzed by several chemometric techniques. Chemometric predictions with sufficiently low errors and collection times to provide distillation control have been made. These predictions demonstrate that a fiber optic based Raman spectroscopy system is feasible.

Free Optics

The scheme referred to as "free optics" is shown in Figure 2-1. [Garrison 90] This is an alpha test version of a commercial Raman analyzer that Perkin-Elmer proposes to sell commercially. It includes a neodymium doped yttrium aluminum garnet (Nd:YAG) [Hecht 87, p. 487] continuous wave (CW)

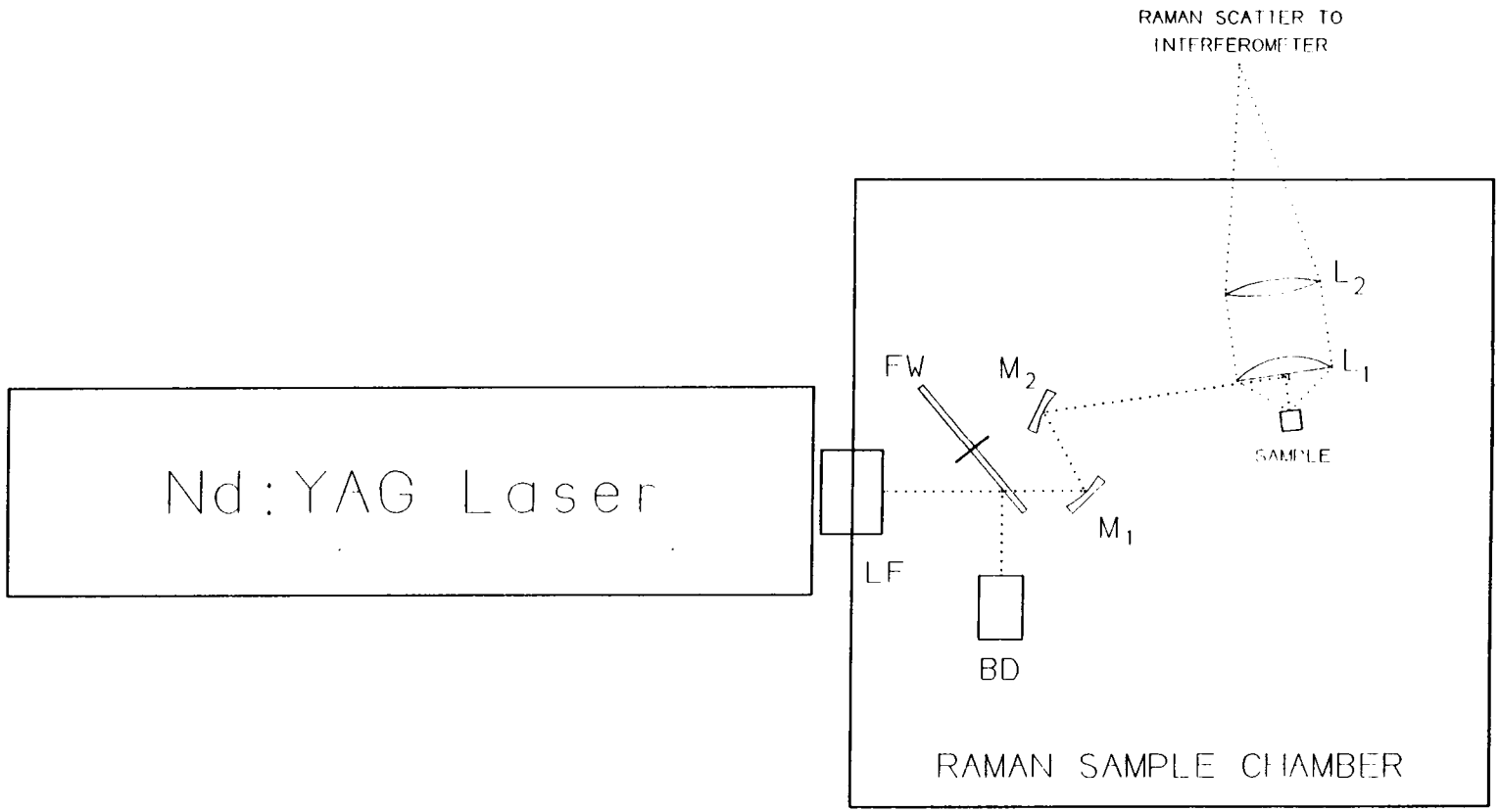


Figure 2-1. Free Optics System

laser that produces about 900 mw at 1.064 microns. The laser is a CVI Laser Corp. Model C95. Laser output is filtered through a line filter (LF) to assure spectral purity. It is attenuated to the desired power level by a neutral density filter wheel (FW), with the unwanted power absorbed by a beam dump (BD). The beam is directed to the sample by mirrors M1 and M2 and a small reflecting prism cemented to lens L1. The sample (contained in a quartz cuvette) scatters the laser light in all directions by the Rayleigh and Raman mechanisms previously described. The light scattered into the solid angle subtended by lens L1 is focused by the lens system L1 and L2 into the Jacquinot stop of a Perkin-Elmer Model 1720 interferometer. The interferometer includes long pass filters to suppress the Rayleigh scattered light, while allowing the longer wavelength Raman bands to pass. The spectral data collected by the interferometer is passed to a Perkin-Elmer Model 7500 computer (a microcomputer based on the 68000 microprocessor) via an IEEE-488 bus. The computer is used for a wide range of operations, such as storing the spectra to disk, plotting, chemometric analysis, and many others. This setup is a typical laboratory Raman analyzer. [Hecht 87, p. 554]

System with Short Fibers

The Perkin-Elmer free optics analyzer is not directly suitable for on-line use since it requires the sample to be contained in a cuvette in the sample chamber. However, the major elements of the system were used in a table-top version of an on-line system as shown in Figure 2-2. Laser output is directed into a 5X microscope objective with a numerical aperture of 0.12. This focuses the collimated laser beam onto the core of an optical fiber. The light is then conveyed by the fiber to a probe whose end is immersed in the liquid sample. Light emerging from the probe is scattered by the sample by the Rayleigh and Raman mechanisms as previously described. A portion of the scattered light is coupled into the receiving fibers of the probe. This light is then conveyed by a bundle of fibers to the interferometer. The light from the fiber bundle is imaged into the Jacquinot stop of the interferometer by a two lens system.

The probe consists of a bundle of receiving fibers surrounding a single exciting fiber. Two probe designs were tested. Both designs are shown in Figure 2-3. The 4-around-1 design is built by EIC Corp. The 6-around-1 design is patented by Dow Chemical Corp. [McLachlan 86] For both probes, the fiber leads are only a few feet long.

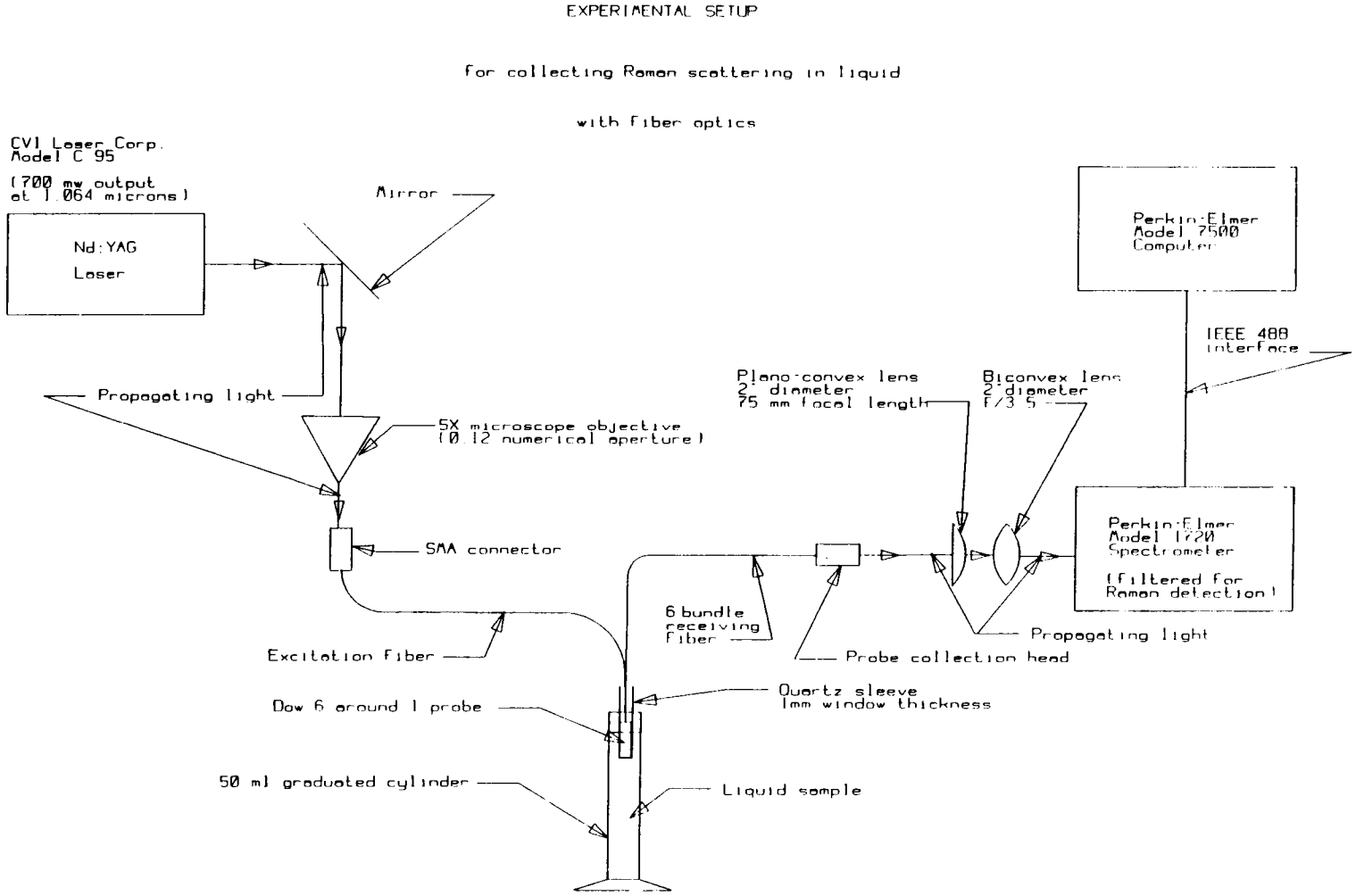
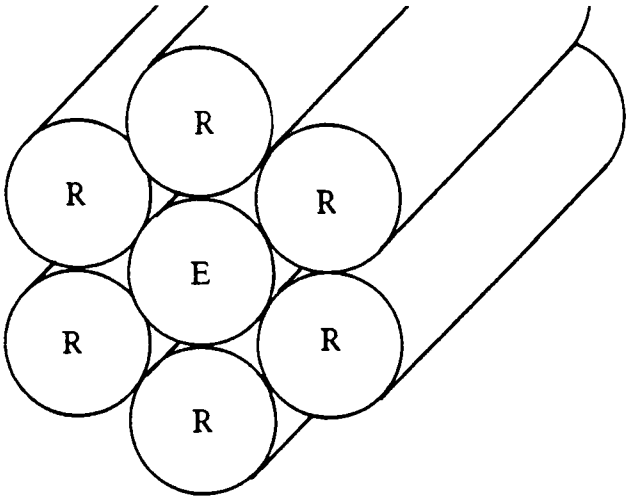
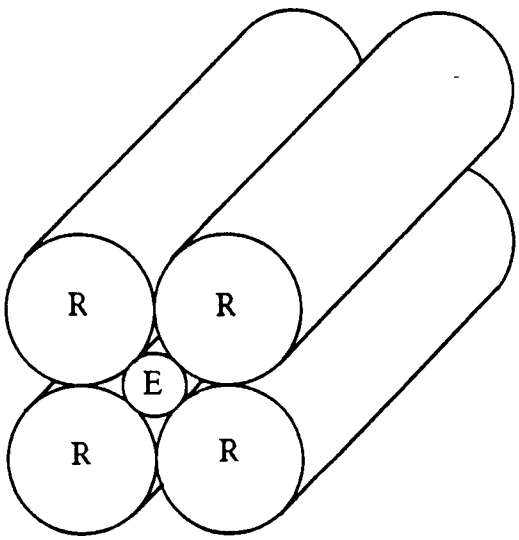


Figure 2-2. Short Fiber System



Dow Design

E - Exciting Fiber
R - Receiving Fiber



EIC Design

Figure 2-3. Fiber Optic Probes

System with Long Fibers

Preliminary testing with long optical fibers was done with the system shown in Figure 2-4. This is similar to the table top scheme shown in Figure 2-2, but it has been modified to accommodate disconnectable long fibers. Laser power is coupled into a long excitation fiber, the other end of which is connected to center fiber of the sample head through a bushing and SMA connectors. The sample head uses the Dow patented six around one probe design. All six receiving fibers of the sample head are connected to the receiving fiber bundle through bushings and SMA connectors. Similar connections are used to connect the receiving bundle to the spectrometer head.

Spectra and Chemometric Predictions

The output of the analyzer is the chemometric prediction of the composition of a chemical mixture. The whole object of this research is to produce chemometric predictions within a desired degree of accuracy. All other spectra, data, and computations are incidental to this purpose. The goodness of a system is judged on how close the chemometric prediction is to the actual composition.

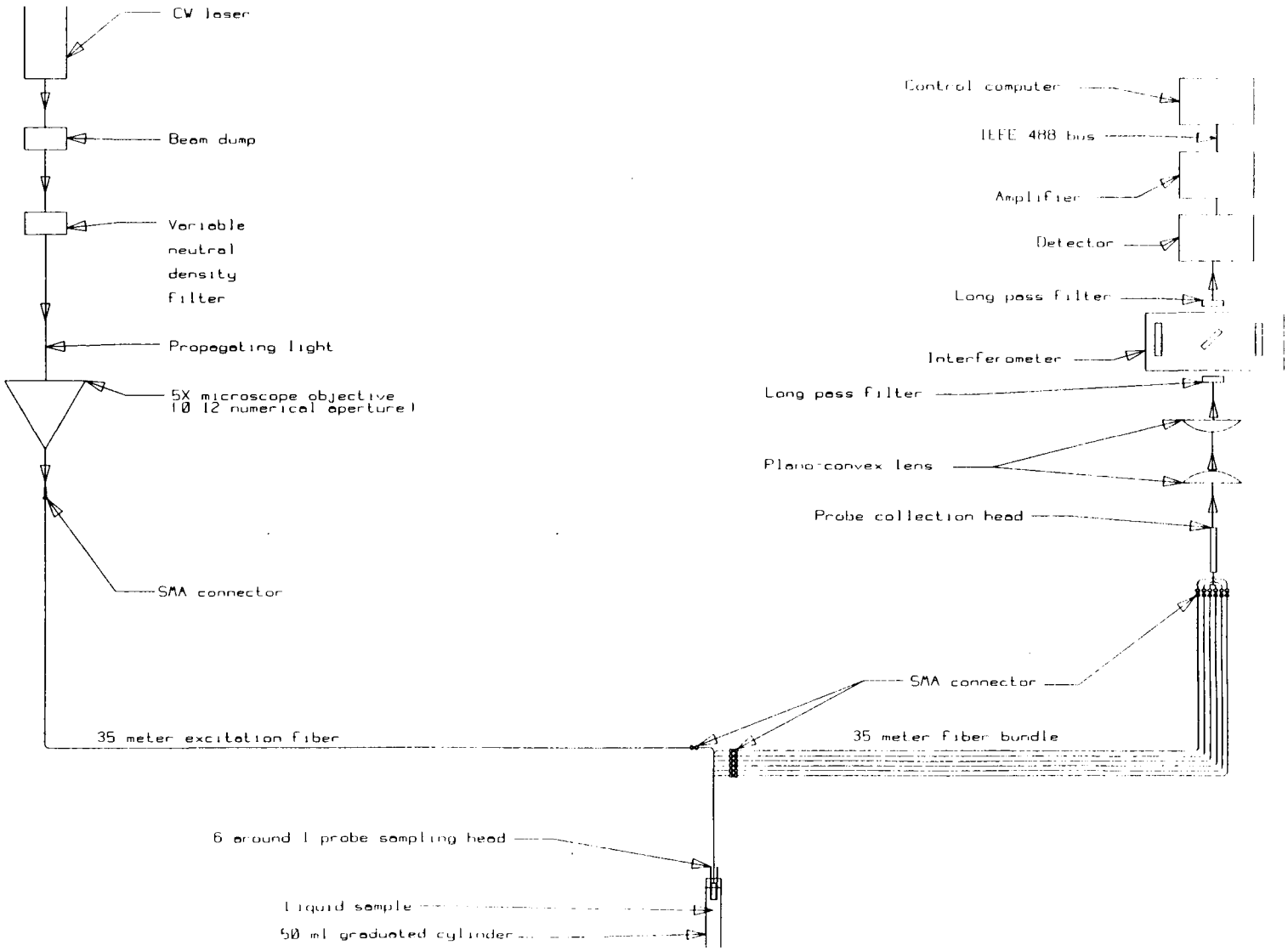


Figure 2-4. Long Fiber Initial Setup

The key finding of this research is that both the short and long fiber based systems produced spectra and chemometric predictions of sufficient quality to control a distillation column. Typical Raman spectra for a variety of chemical samples, collected with several different systems, are shown in Appendix C. Both the short and long fiber based systems produced chemometric predictions with an error of less than two percent RMS, the accuracy needed for column control.

The accuracy of the short fiber based system in predicting the composition of distillation mixtures is shown in Figure 2-5. The Eastman column distills an acetic extract mixture. The UT Chemical Engineering column distills an isopropanol mixture. The Hercules process monitors a cumene mixture. See Appendix D Section 1 for chemical details of the mixtures. For each mixture, four samples were analyzed with the system depicted in Figure 2-2. The RMS errors using the 1220-620 band spectrum are indicated in the bar graph. Two analyses were made for each sample, one with a collection time of 2.2 minutes (16 scans), and one with a collection time of 8.7 minutes (64 scans). Detailed analysis of the accuracy of the chemometric predictions is included in Appendix D.

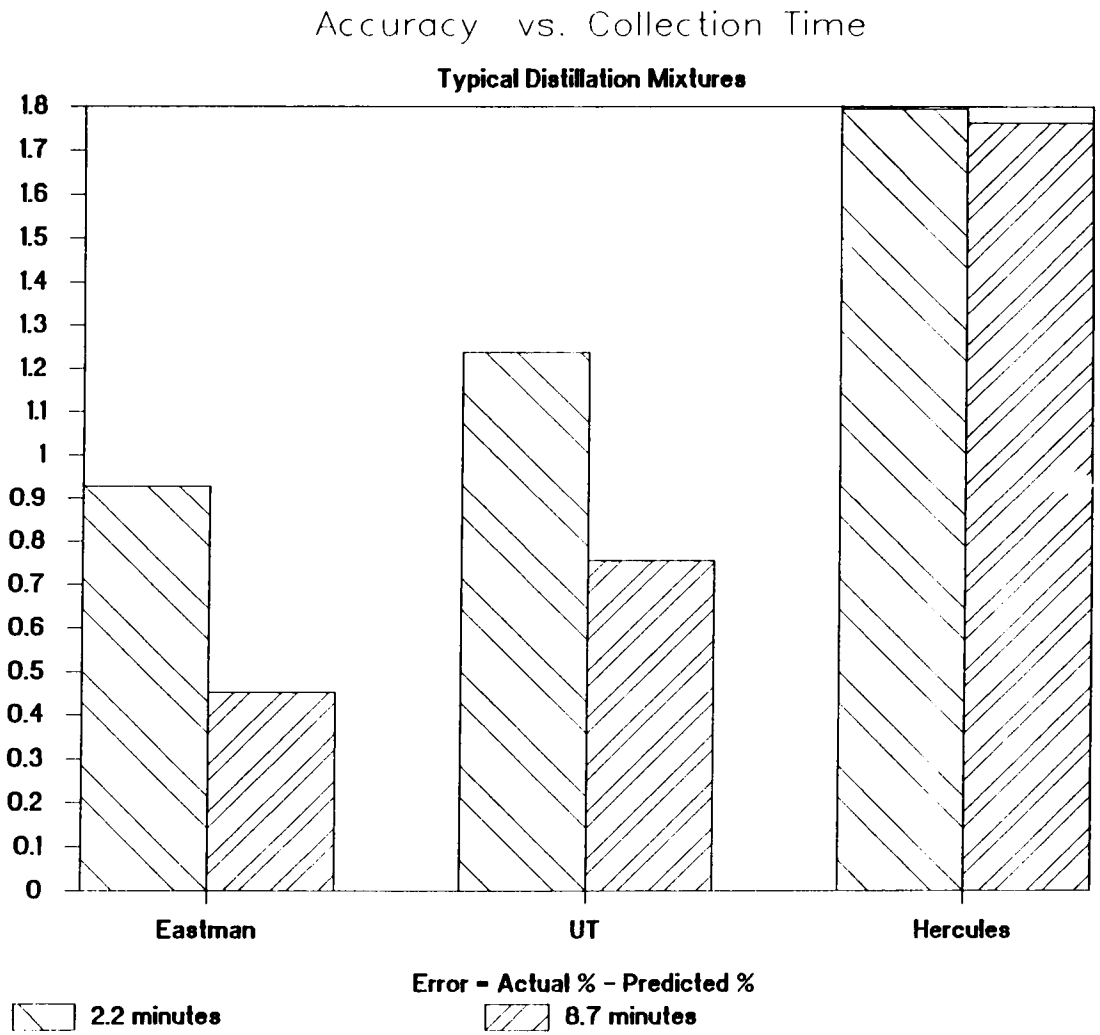


Figure 2-5. Accuracy of Chemometric Predictions

Note that for the long fiber system, spectra were only taken for a collection time of 8.7 minutes (64 scans). Available laser output was only 500 mw and spectra taken for short collection times would have been too noisy for accurate chemometrics. However, the chemometric results for 64 scans indicate that similar results will be obtained in 2.2 minutes (16 scans) when a full watt of laser power becomes available.

Chemometric Analysis Algorithms

Chemometric analysis can be viewed as a vector transformation. The input is a long vector (dozens or hundreds of elements) representing spectral intensities. The output is a short vector (as many elements as there are components in the mixture) representing concentrations of each component. Several matrices transform the spectral intensity vector into a concentration vector.

The problem of obtaining the transformation matrices, or "training" the system, consists of using a number of known pairs of vectors (that is, a known sets of concentrations that correspond to known spectra) to deduce the transformation. This can be done with various regression algorithms.

The two algorithms considered for Raman spectra were multiple linear regression (MLR) and partial least squares regression (PLS). [Martens pp. 63-64, 112-163] [Massart 88] Both are implemented in commercial software packages. MLR analyses reported in Appendix D use the CIRCOM package furnished with the Perkin-Elmer equipment. PLS analysis of the Eastman mixture reported in Appendix D was performed by Sprouse Scientific Systems, using the Sprouse commercial package. [Sprouse 89]

The chemometrics algorithm implemented in the PC based control system uses factorization and multiple linear regression. [Fredericks 85b]. The spectra are represented in a matrix **D** in which each column is a vector of spectral intensities for a training sample. The information in **D** is compressed by forming $\mathbf{Z} = \mathbf{D}^T\mathbf{D}$, the product moment matrix. The eigenvectors of **Z** form the loading matrix **L**, and the eigenvalues of **Z** are the diagonal elements of the matrix $[\lambda]$. The eigenspectra, or factors, **F**, are the set of vectors, a linear combination of which will form any of the training spectra. The factorization is $\mathbf{F} = \mathbf{DL}$. The largest eigenvalue represents the most information and the least noise. The smallest eigenvalue represents the least information and the most noise. In view of this, the matrices **F**, **L**, and $[\lambda]$ are arranged with the eigenvalues and corresponding columns progressing from greatest to smallest

left to right. Several of the rightmost columns can be eliminated from the analysis, since they represent error. [Malinowski 77a]

The regression is based on a reduced and augmented loading matrix, $\mathbf{L}_2$. This matrix is formed by eliminating the noisy columns from $\mathbf{L}$ and augmenting with column of 1's on the left. The regression matrix is $\beta = (\mathbf{L}_2^T\mathbf{L}_2)^{-1}\mathbf{L}_2^T\mathbf{P}$, where $\mathbf{P}$ is the matrix of composition vectors of the training samples.

Analysis of an unknown spectrum is a three step process. First, the loading is formed. The loading is the row matrix $\mathbf{uL}_2 = [\lambda]^{-1}\mathbf{F}^T\mathbf{U}$, where $\mathbf{U}$ is the intensity vector of the unknown spectrum. The loadings have varying amounts of correlation with the concentrations in the training samples. By eliminating those elements from the loading whose correlation with the concentration is less than 0.1, the accuracy of the analysis can be improved. Since the correlations between the loading terms and the concentrations are different for each component of the mixture, the process must be repeated for each one. The result is a loading matrix, $\mathbf{uL}_3$, with as many columns as $\mathbf{uL}_2$ and as many rows as there are components in the mixtures used as training samples. The composition of the unknown is $\mathbf{p} = \beta^T\mathbf{uL}_3$.

Partial least squares is a similar technique, but does not presume a linear relationship between a spectrum and a property. [Martens pp. 118-119] There is no clear indication that PLS is a superior performer to MLR. For the Eastman mixture, MLR produced consistently lower errors. For petroleum mixtures, Seasholtz and his associates found that PLS gave consistently better results than MLR. [Seasholtz 89]

Impact of These Experiments on Hardware Specification

The excitation wavelength for these experiments was 1.064 microns. The spectra shown in Appendix C are satisfactory despite the fact that Raman scattering is weak in the infrared. Raman intensity decreases with the fourth power of excitation wavelength. [Hecht 87, p. 294] Visible light is not suitable for Raman spectroscopy of distillation mixtures due to fluorescence. [Garrison 89, pp. 26-44] A number of other researchers have reported successful results at 1.064 microns. [Seasholtz 89] [Everall 89] At the current level of laser and detector technology, 1.064 microns seems to be the most practical wavelength for excitation for Raman spectroscopy.

A resolution of two wave numbers is required for a general purpose instrument. The system must be able to distinguish components with a low concentration from similar components having a high concentration. High resolution is required to prevent the large spectral feature from swamping the small one.

For example, in the cumene oxidate mixture, the system must be able to distinguish between the acetophenone feature at 737 wave numbers and the cumene feature at 746 wave numbers (Figure 2-6). It would appear that a nine wave number resolution should resolve the features. However, the cumene is at a much higher concentration than the acetophenone, and both features have a finite width. In Figure 2-7, spectra of cumene oxidate taken at two and four wave number resolution are overlaid. The small shoulder in the two wave number spectrum is the acetophenone feature. In the four wave number spectrum, it vanishes completely, having been swamped by the cumene feature.

In addition to high resolution, the system requires high bandwidth. 600 wave number bandwidth was required to obtain the best chemometric accuracy in the short fiber system. Since the system must operate with high resolution, high bandwidth, and high throughput, all at the same time, a

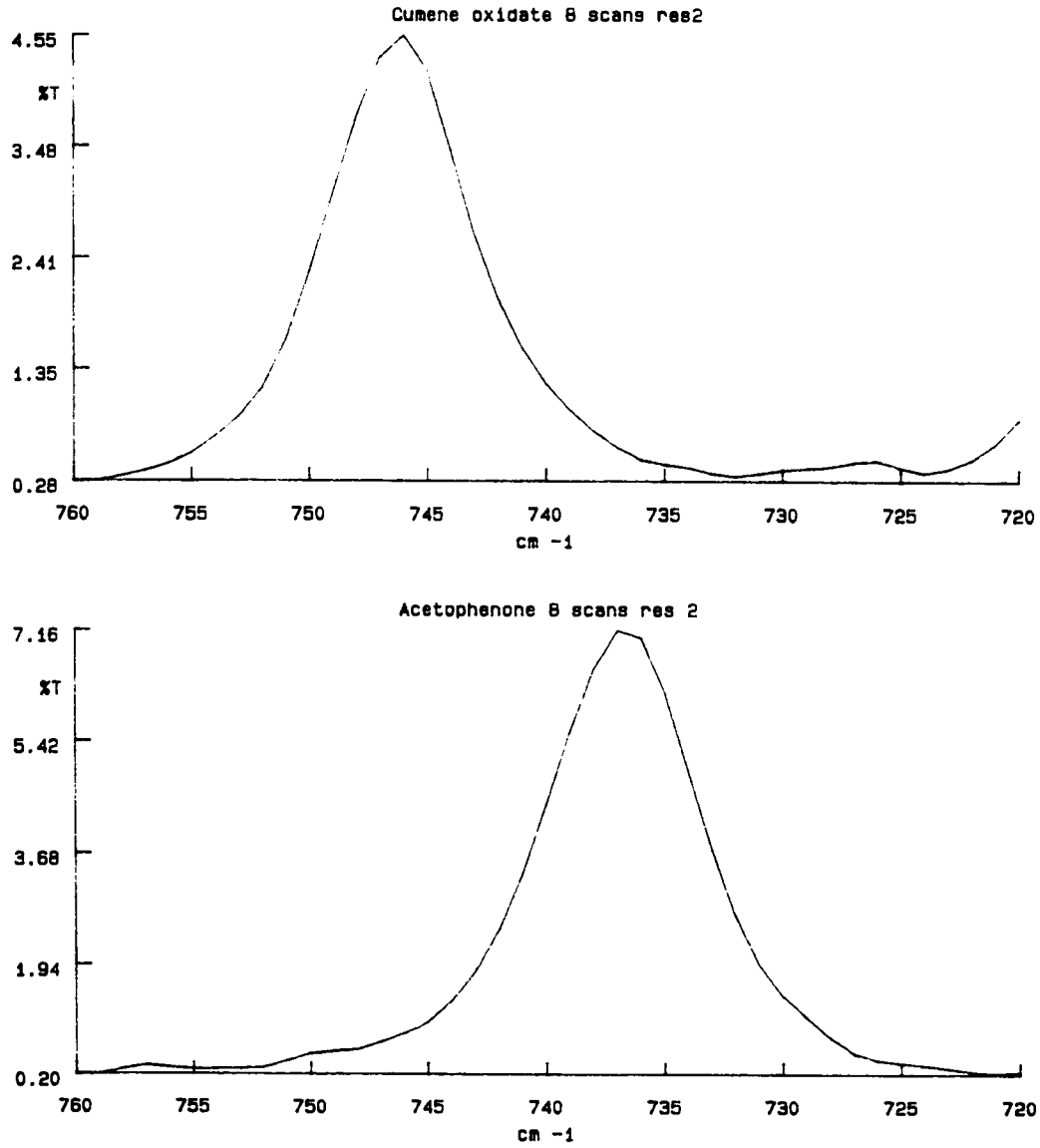


Figure 2-6. Cumene Oxidate and Acetophenone

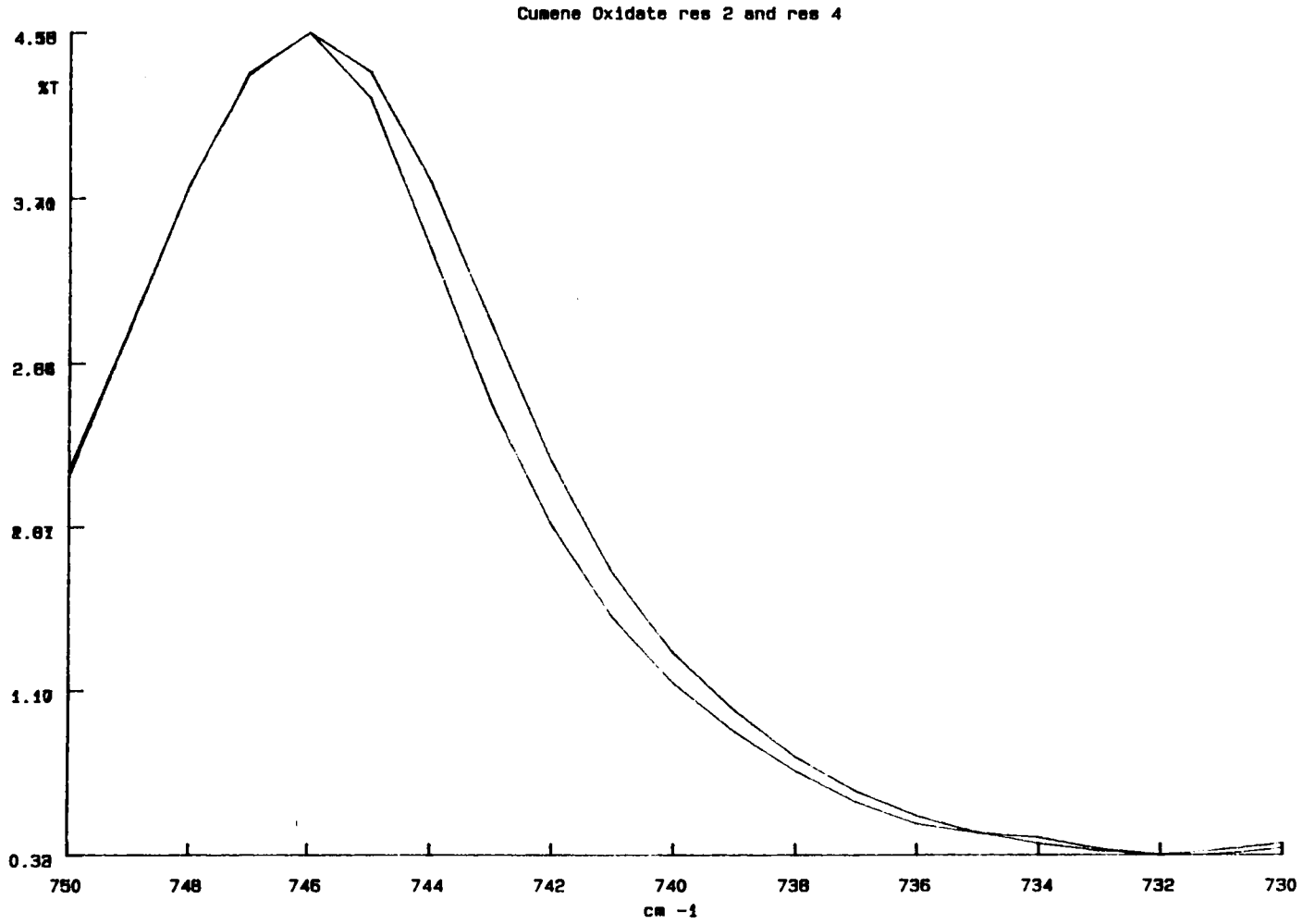


Figure 2-7. Effect of Resolution

grating spectrometer will not meet the requirement. This limitation is consistent with the theory of spectrometers. [Bell 75, pp. 23-25] The main advantage of a grating spectrometer is that no optical filtering is needed to eliminate the Rayleigh line. However, this has not been a problem in these experiments. All the interesting spectral features of the mixtures of interest have a Raman offset of more than 600 wave numbers from the excitation. They are not attenuated significantly by the long pass filter in the interferometer system.

CHAPTER 3

DISCOVERIES MADE DURING SHORT FIBER EXPERIMENTS

Summary of Discoveries

Incidental to producing the chemometric results indicated in Chapter 2, a number of interesting experimental discoveries were made. The probe must be protected from being attacked by the sample, but the performance cost of the protection is not great. The region in which useful Raman energy is generated extends approximately 100 mm from the probe tip. The Dow probe has stronger coupling than the EIC probe in the band of interest for chemometrics. The isopropanol line height varies non-linearly with the isopropanol concentration in an isopropanol water mixture. Noise in the observed spectrum is independent of the excitation power. Over a span of hours, the height of the Raman features tends to drift.

Window

It is not desirable to expose a fiber optic probe directly into a liquid sample. The sample may (and quite often does) damage the probe. A quartz window 1 mm thick at

the end of the probe is used for the short fiber experiments.

Both the Dow and EIC probes are affected by direct immersion in the sample. The fibers in the EIC probe are held in place with an epoxy that softens in acetic acid. The Dow probe is held in a steel shell that rusted after repeated exposure to various samples. To avoid further probe damage, the use of windows was investigated.

Interposing a window between the probe end and the sample adds loss to the system. Losses were measured with the Dow probe and found to be less than one dB. Results are listed in Table 3-1.

Table 3-1. Attenuation Through Quartz Window

	Bare Probe:		Through Window:		Degradation:	
Sample	Signal %T	S/N Ratio dB	Signal %T	S/N Ratio dB	Signal dB	S/N Ratio dB
Benzene	20.48	21.65	17.72	20.72	0.63	0.93
Isopropanol	2.04	12.68	1.7	11.89	0.79	0.79
Acetic Extract	0.65	8.59	0.59	7.71	0.42	0.88

There are other possible window materials besides quartz. Diamond and sapphire windows are often used for this kind of application. However, both have a high index of refraction. Diamond has an index of refraction of 2.4. [DeBeers 89] Sapphire has an index of refraction of 1.75. [Melles Griot 85, p. 65] On the other hand, indices of quartz, silica, and most of the samples range between 1.35 and 1.62. [CRC 81 p. C-72, C-78, C-142, C-244, E-380-1], [Melles Griot 85, p. 56] Hence, there is very little reflection loss in the present system. If a high index window is placed between the fiber and the sample, it would add a substantial reflection loss. [Hecht 87, pp. 81-87] The exciting signal would pass through two interfaces, and the scattered signal would pass back through the two interfaces. All these reflections would accumulate and dramatically reduce the intensity of the detected signal.

Care must be taken to assure that the probe is tightly seated to the window. If there is an air gap between the end of the probe and the window, additional attenuation is introduced due to reflection loss of four new air-glass interfaces. Also, the Raman scattering in the excitation fiber is reflected from the window and coupled into the receiving fibers.

Where does the Raman Scattering Occur?

For the Dow probe, the Raman scattering that is detected by the probe occurs at a distance of 10 to 100 mm from the end of the probe. The experimental demonstration of this is recounted in Appendix E. Since the Raman scattering is much further than 1 mm from the end of the probe, it is not surprising that a 1 mm window on the end of the probe does not significantly interfere with the its operation.

Comparison of Dow and EIC Probes

A comparison of the spectra in Appendix C leads to an unexpected discovery. For cumene, acetic extract, and isopropanol, the free optics and EIC probe spectra show features in the 1500-1400 wave number band. However, these features are not present in the spectra taken with the Dow probe. This suggests that the Dow probe filters out features in this band.

This was tested as follows. A spectrum (Figure 3-1) for "white" light was taken directly at the input of the interferometer. A spectrum (Figure 3-2) of the same light was taken through the Dow probe. Finally, the response of the Dow probe (Figure 3-3) was calculated by dividing the data in Figure 3-2 by the data in Figure 3-1. This shows

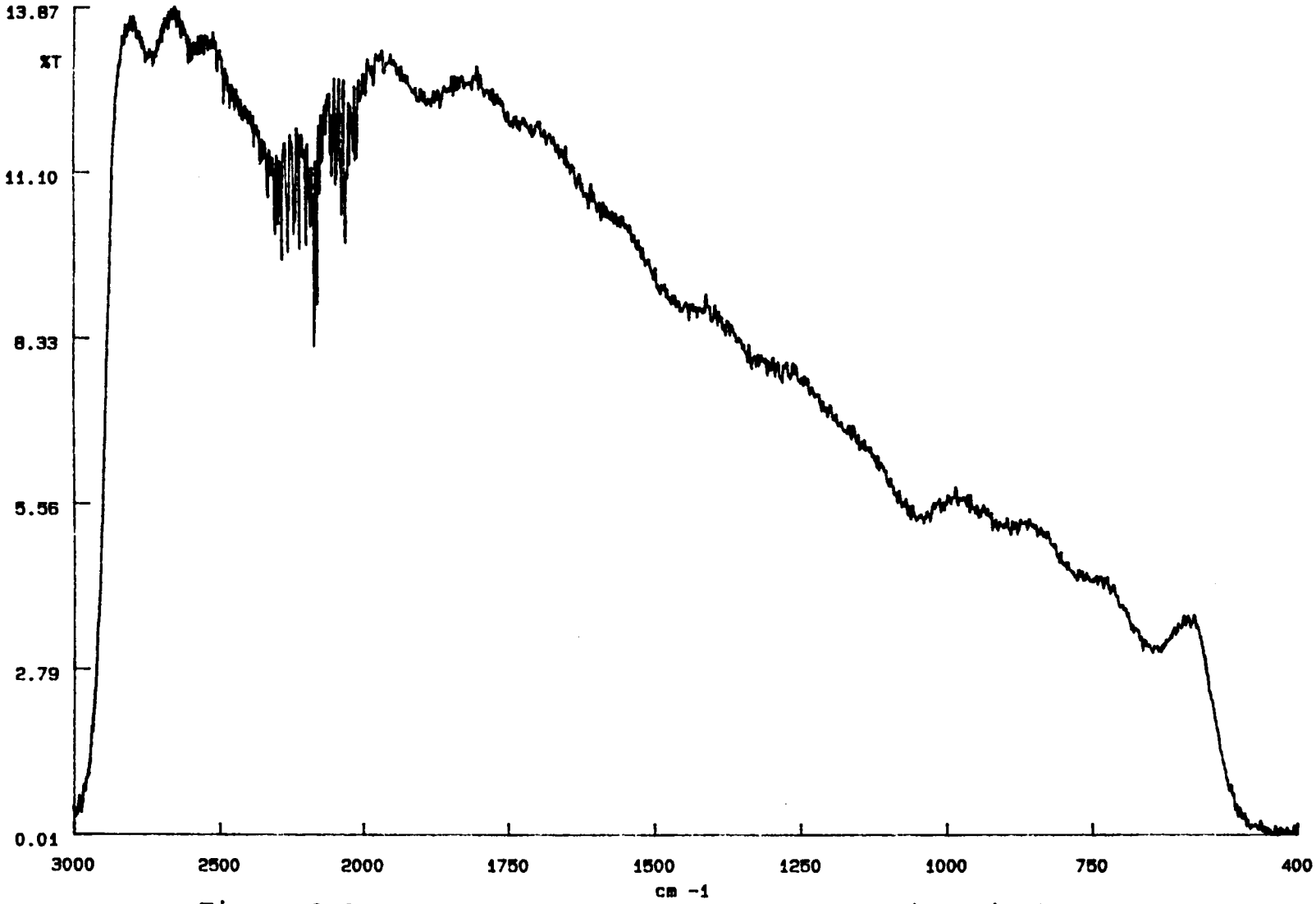


Figure 3-1. Response of Interferometer to White Light

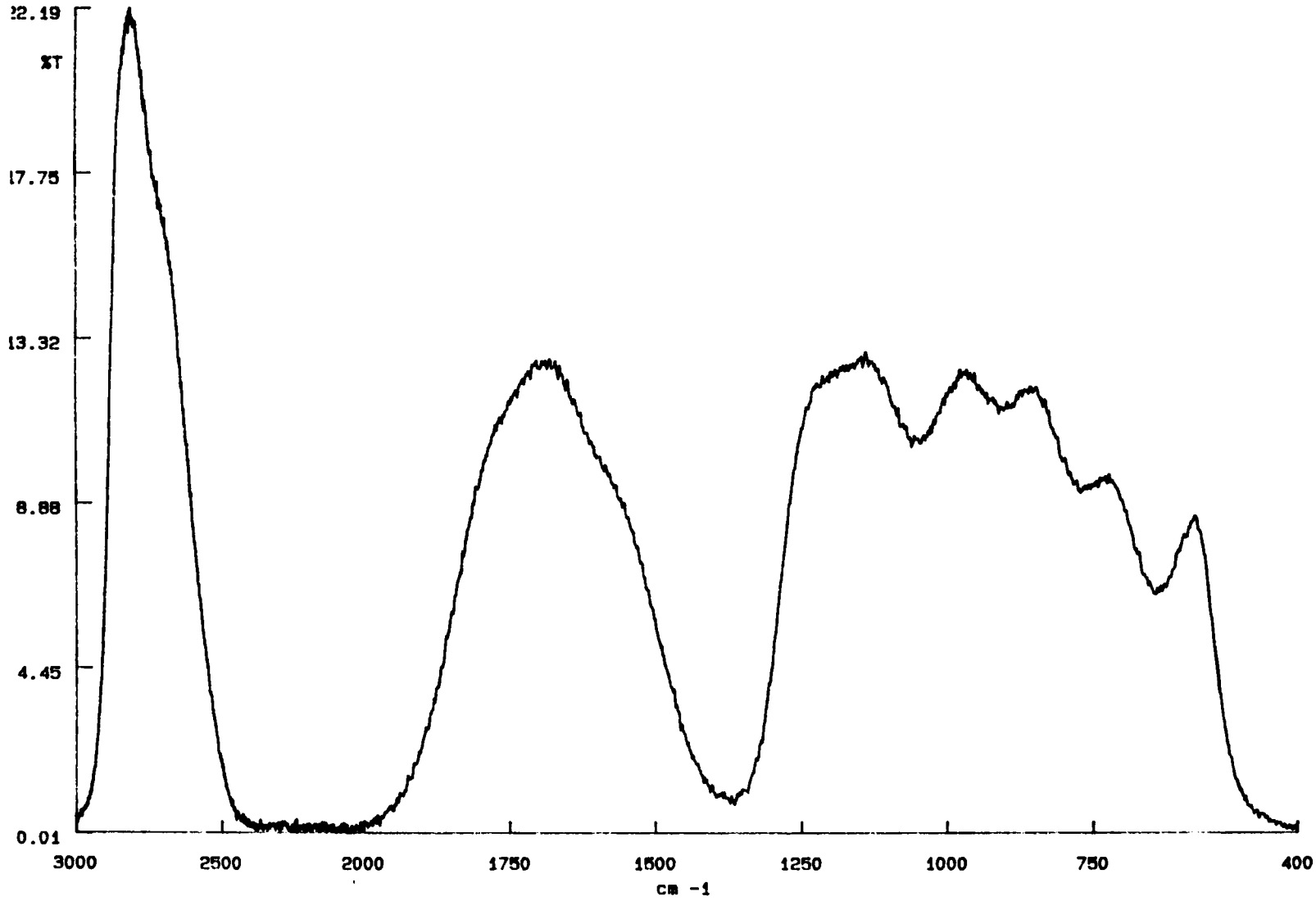


Figure 3-2. White Light Response of Dow Probe

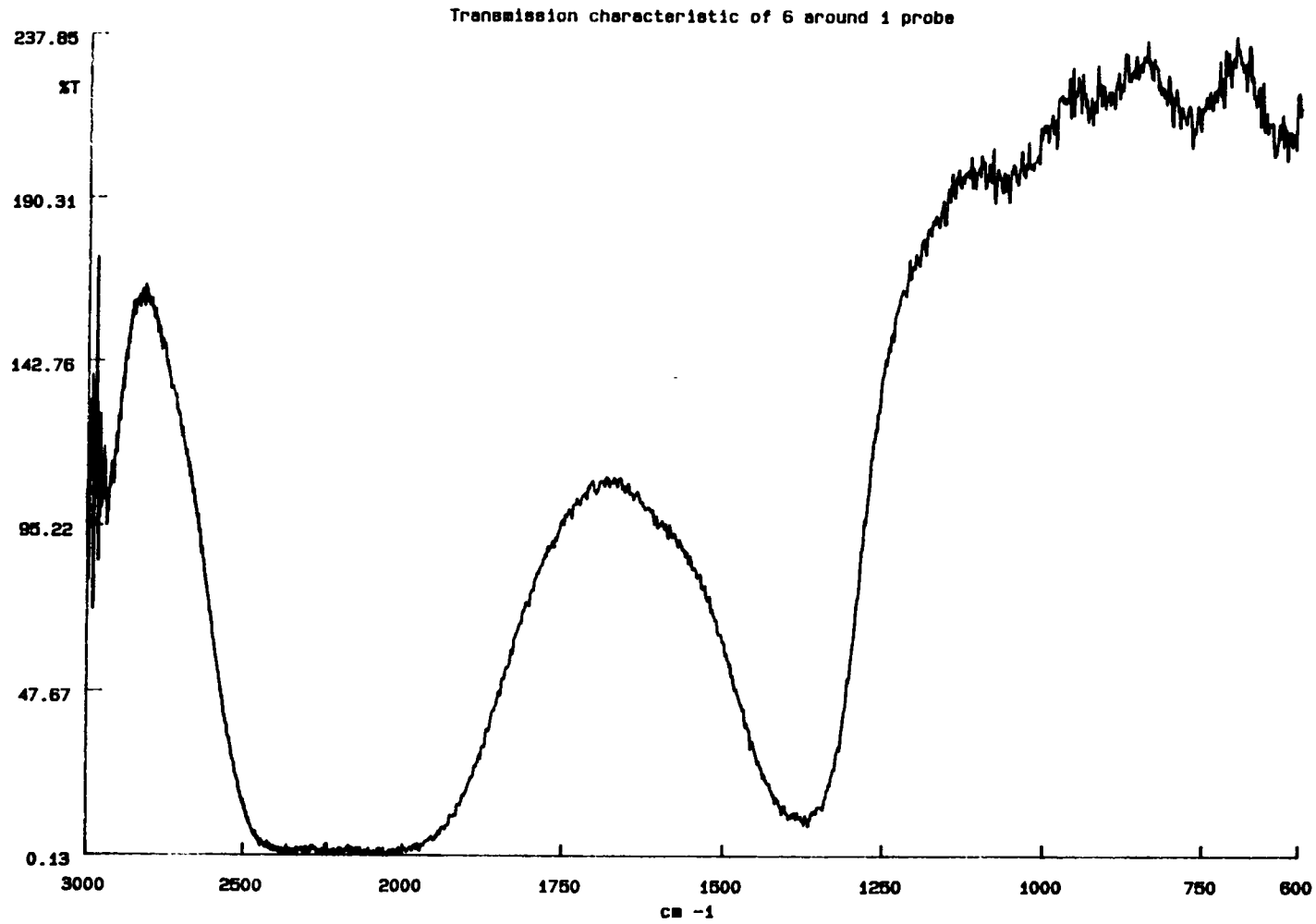


Figure 3-3. Wavelength Response of Dow Probe

that there is a cutoff in the 1400 wave number area and an even more dramatic one in the 2500-2000 wave number area. It is fortunate that the probe has good transmission in the 1200-600 wave number band, the region containing most of the information used by the chemometric software.

A similar study was made of the EIC probe. An interferometer base line was taken (Figure 3-4). A spectrum of the white light through the EIC probe was taken (Figure 3-5). The data in Figure 3-5 were divided by the data in Figure 3-4. The resulting normalized response is shown in Figure 3-6. This shows that the EIC probe has essentially flat response throughout the range of interest.

Despite its peculiar frequency response, the Dow probe is a more consistent performer than the EIC probe. Spectra were run with both probes, and using free optics, for three different samples. Excitation to the sample was about 500 mw for all cases. The probe was in the quartz sleeve for all fiber optic measurements. Peak intensities are listed in Table 3-2.

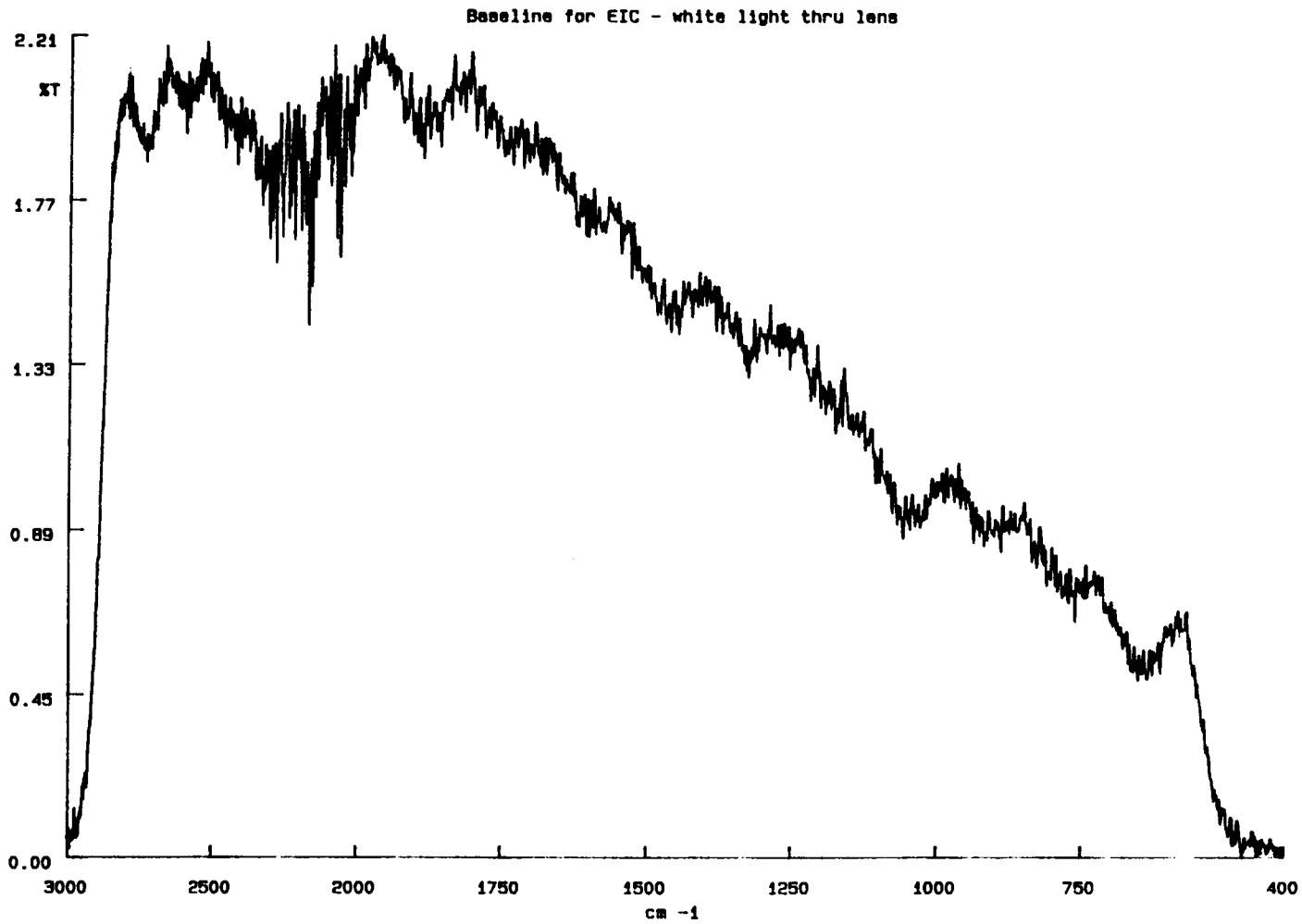


Figure 3-4. Baseline for EIC Response

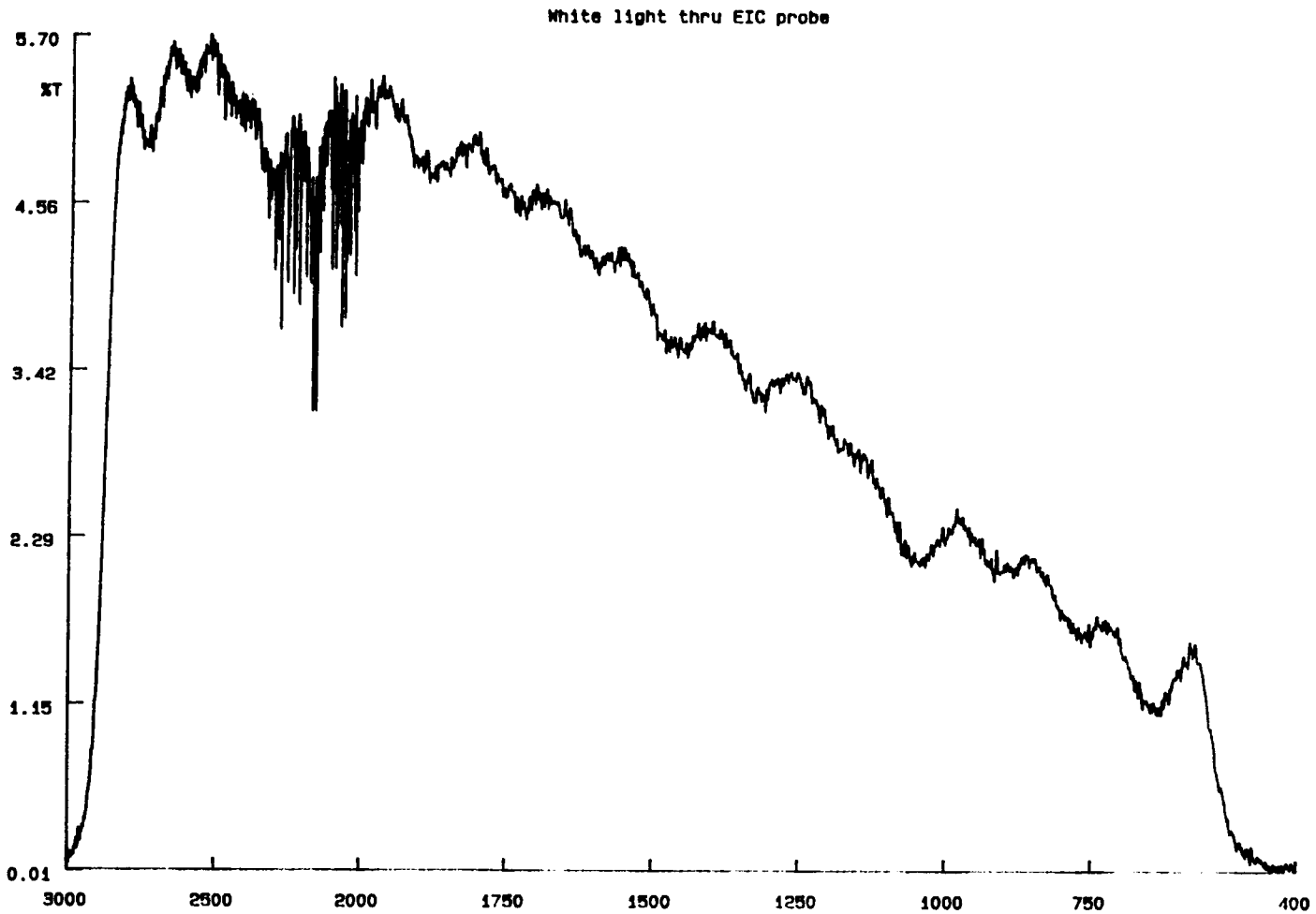


Figure 3-5. White Light Response of EIC Probe

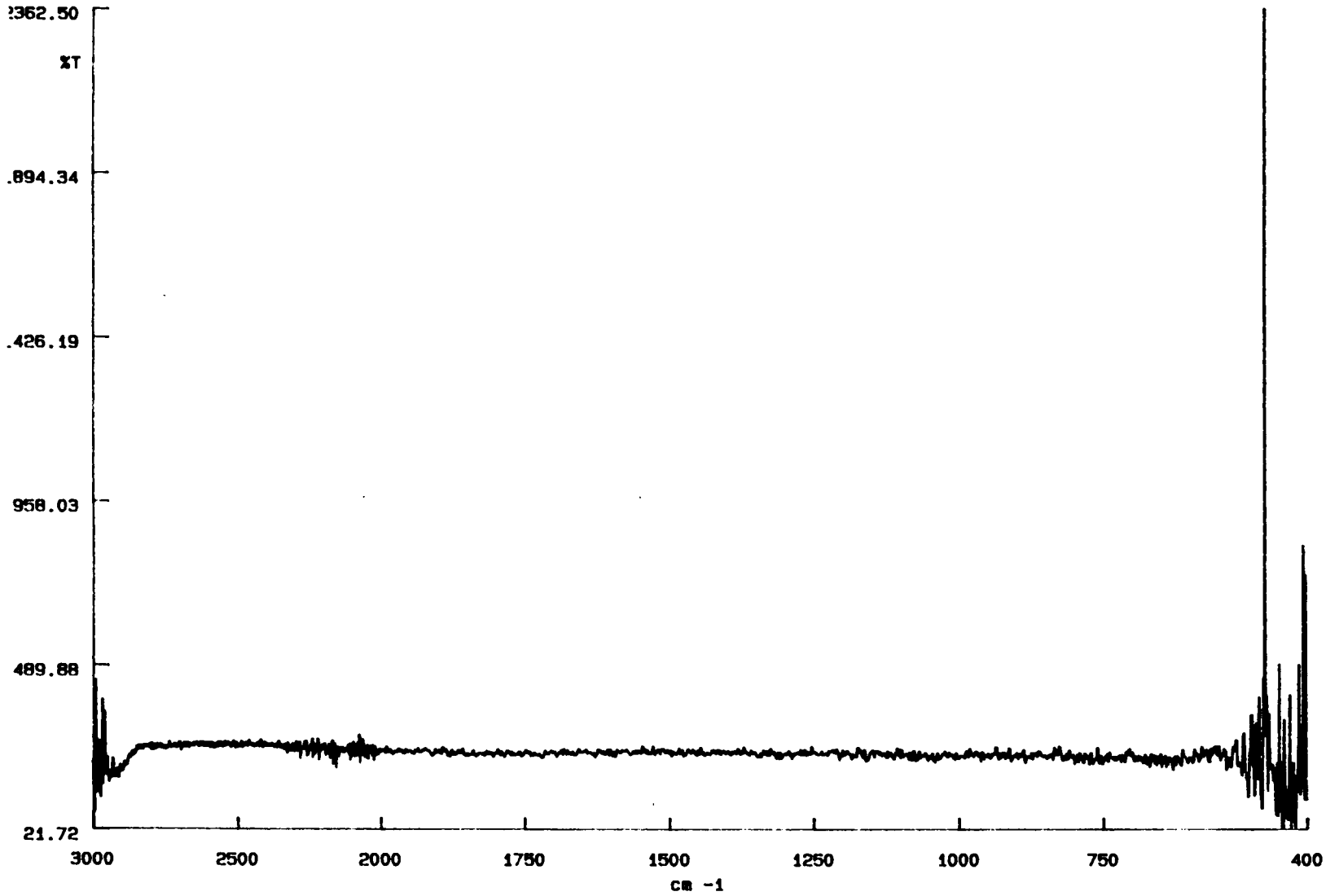


Figure 3-6. Wavelength Response of EIC Probe

Table 3-2. Comparative Raman Peak Values

Sample	Peak %T: EIC Probe	Dow Probe	Free Optics	Dow vs. EIC	Free vs. Dow
Acetic Extract	0.18	0.62	1.70	5.37 dB	4.38 dB
Cumene	1.80	3.18	6.51	5.72 dB	3.11 dB
Isopropanol	0.53	1.70	3.35	5.06 dB	2.95 dB

For the purpose of chemometric analysis, a system that gives five dB higher signal level in the 1220-620 wave number band and poor response everywhere else is more valuable than a system that gives a low flat response across a wide band. Thus, the Dow probe is the more desirable of the two.

One other conclusion can be drawn from the data above. Although it was not done at the same time, measurements on benzene at 520 mw excitation with free optics produce a peak intensity of 61.98 %T, 650 mw excitation with the Dow probe produce a peak intensity of 17.72 %T. If the Dow probe reading is normalized to 520 mw the peak would be 14.17 %T. Thus the free optics produces a signal 6.4 dB stronger than the Dow probe for benzene. Taking this result and the data tabulated above, it is apparent that the performance cost of going from free optics to a Dow probe is in the range of 3 to 6 dB.

Non-linearity of Isopropanol

For most chemicals, the intensity of the Raman features is linearly proportional to the concentration of the component. This proposition is the basis for using multiple linear regression to do quantitative analysis. However, as seen in Figure 3-7, this is not true for the isopropanol water mixture. The isopropanol peak is a slightly non-linear function of the concentration of isopropanol.

The explanation for this effect is as follows: In most mixtures, the components act independently of each other. Each component scatters light in proportion to the number of scatterers present, leading to the linear effect described above. However, both the isopropanol and water molecules have a hydroxyl radical. This causes some interaction in the scattering behavior of the two molecules, and leads to the non-linear response. [Garrison 89, p. 45]

Limits on System Performance

The detector/amplifier system used with the Perkin-Elmer 1720 interferometer consists of an indium gallium arsenide detector and a transimpedance amplifier, both contained in a Dewar flask and cooled with liquid nitrogen. The literature says that, for an infrared interferometer,

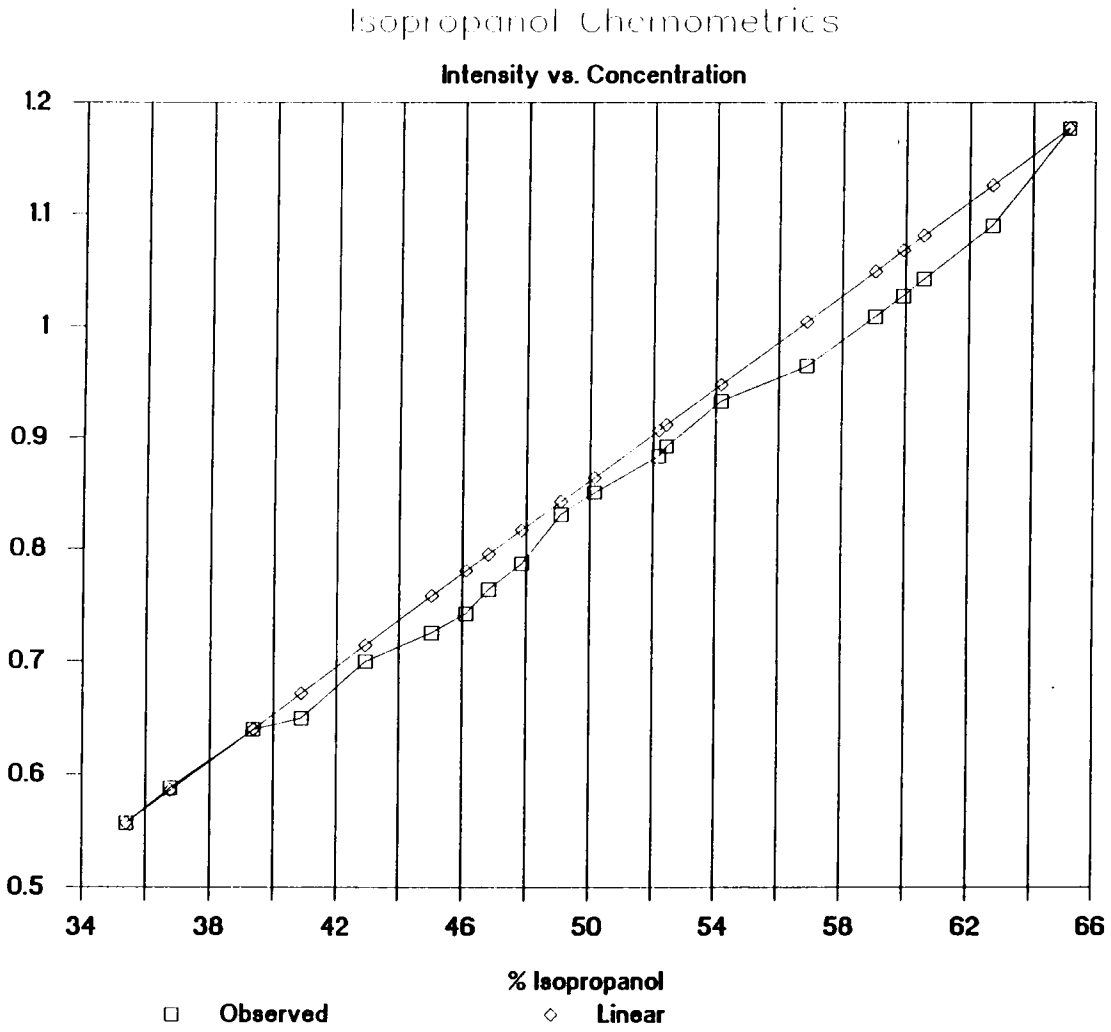


Figure 3-7. Non-Linearity of Isopropanol

system performance is ultimately limited by detector/amplifier noise. [Bell 75, pp. 23-25] The limitation arises from the fact that in addition to detecting a signal and increasing its strength, a detector/amplifier system adds some artifacts of its own. If the amplifier is presented with a very weak input signal, it may be impossible to distinguish the signal from the background artifacts. [Millman 67, pp. 472-479] These artifacts can be generated by deterministic means (spurious signals) or by random means (noise).

Spurious signals can be generated by various mechanisms. The most notable are parasitic oscillations, power frequency pickup, and microphonics. Parasitic oscillations arise from the fact that stray capacitances, inductances, and resistances lead to signal paths not anticipated in the system design. These can lead to feedback that can cause the amplifier to oscillate. This can be remedied by using a physical layout that minimizes stray component effects, and by using shielding to break up the feedback paths. Power frequency pickup arises from the fact that nearby power system wiring emits an electromagnetic field that induces a power frequency signal into an unshielded amplifier. In addition, if the amplifier and power supply are not properly grounded, then power frequency energy can be coupled into the amplifier. Microphonics are

caused by the fact that sound induces physical vibrations in the components of the amplifier. The vibrations cause the electrical properties of the components to vary, and this variation is superimposed on the amplified signal. Observations done with an oscilloscope and a spectrum analyzer on the Perkin-Elmer detector/amplifier system indicate that, when properly connected, all three spurious signal effects are insignificant.

Noise is also generated by several mechanisms. The two most notable noise mechanisms are Johnson noise and shot noise. Johnson noise is a random fluctuation in signal due to thermal agitation of the electrons carrying the signal. The RMS value of Johnson noise voltage is $V^2 = 4KTR\Delta f$, where K is Boltzmann's constant, T is the absolute temperature of the device, R is the resistance of the device, and Δf is the bandwidth of the signal being amplified. Note that noise depends on bandwidth, but not absolute frequency. Hence, in the power spectrum of Johnson noise, the power in the noise is essentially independent of frequency. It is white noise. Shot noise arises from the fact that electric current is the flow of a finite number of electrons, each carrying a discrete charge. The RMS value of the noise current is $I^2 = 2eI_s\Delta f$, where e is the charge of an electron, I_s is the signal current, and Δf is the bandwidth. Again the signal is white noise, its energy being independent of frequency. It

is notable that Johnson noise is independent of applied signal strength, but shot noise increases with the square root of signal strength.

Several experiments reveal the character of the noise that limits the performance of the Perkin-Elmer detector/amplifier system. In one experiment, spectra were taken by free optics for benzene samples for a range of excitations from 50 to 500 mw. The benzene peak varied linearly in the range, but the noise peak stayed approximately constant. Noise did not increase with increasing excitation.

Another experiment observed the detector spectrum with all light sources cut off. At 16 scans the peak was 0.13. At 64 scans, the peak was 0.08 as shown in Figure 3-8. These levels are consistent with spectra of many different samples taken at 16 and 64 scans. The experiment bears out another well known principle. When making measurements in a noisy environment, the average of many measurements causes the noise to cancel out. The amount of noise measured goes down as the square root of the ratio of the number of measurements taken. [Kreyszig 67, pp. 790-798] A result based on 64 scans has 4 times as many readings as a result based on 16 scans. Thus, it is expected that the result of 64 scans should contain half as much noise as the result of

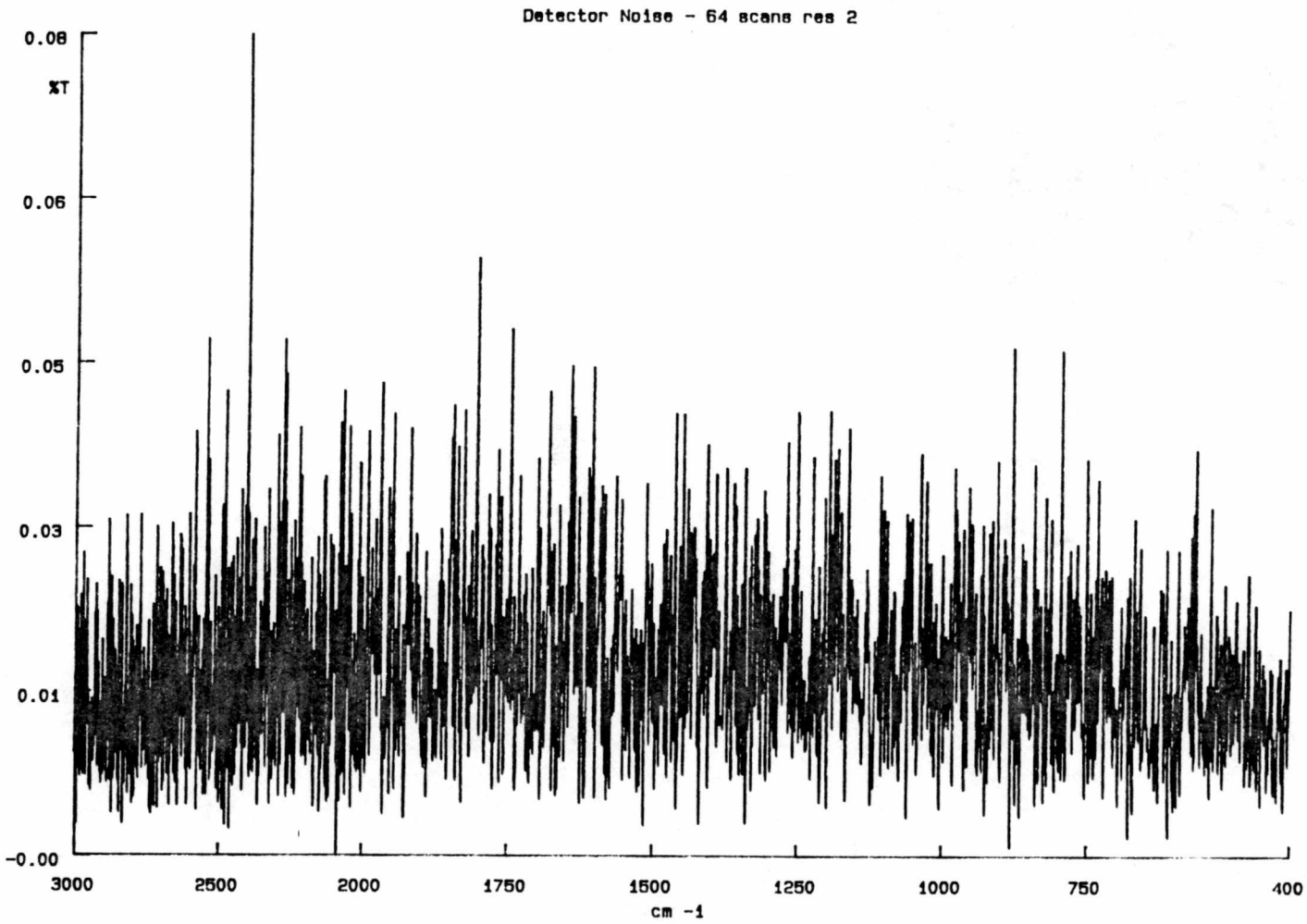


Figure 3-8. Detector Noise

16 scans. The above experiment shows this to be approximately true.

The foregoing observation indicates why spectra for chemometrics were taken at both 16 and 64 scans. The signal to noise ratio of the 64 scan spectrum is 3 dB better than the signal to noise ratio of the 16 scan spectrum. The goodness of the chemometric prediction depends on the signal to noise ratio rather than the absolute signal level. Thus, while the 16 scan result is an indicator of system performance, the 64 scan result is a predictor of system performance if a 3 dB improvement in performance can be found.

The variation of noise with detector temperature was observed for acetic acid spectra. Spectra were taken for the same acetic acid sample with the detector at room temperature and with the detector cooled with liquid nitrogen. Both spectra were the average of 128 scans. Noise is defined as the peak intensity in the 2600-2000 wave number band. For the hot detector the noise was 0.18 %T. For the cold detector, the noise was 0.03 %T. In this experiment, cooling the detector/amplifier system from room temperature to liquid nitrogen temperature reduced the noise by a factor of six.

These observations show that the system performance is limited by noise. The noise is independent of the strength of the applied signal. The noise in spectra is reduced by increasing the data collection times. The noise is strongly dependent on detector temperature. These facts indicate that the limiting mechanism is Johnson noise. In a mathematical analysis of the system made by Dr. M. J. Roberts, the limiting mechanism was further narrowed to the Johnson noise arising from the first stage of the amplifier. [Roberts 89]

Raman Scattering by Water

Comparison of Figures 3-8 and 3-9 bear out the idea that water is a weak Raman scatterer. In fact the Raman spectrum for water cannot be distinguished from noise. The 800-600 wave number band peaks of Figure 3-9 appear to be glass artifacts from the vessel containing the water. Similar artifacts can be seen in Figure 3-10, the spectrum for an empty quartz cuvette.

Line Height Drift

One other surprising observation was a large amount of drift in the peak intensity of the spectrum over a span of time. Figure 3-11 shows the variation in peak intensity of a benzene spectrum over a span of 10.5 hours. A spectrum was

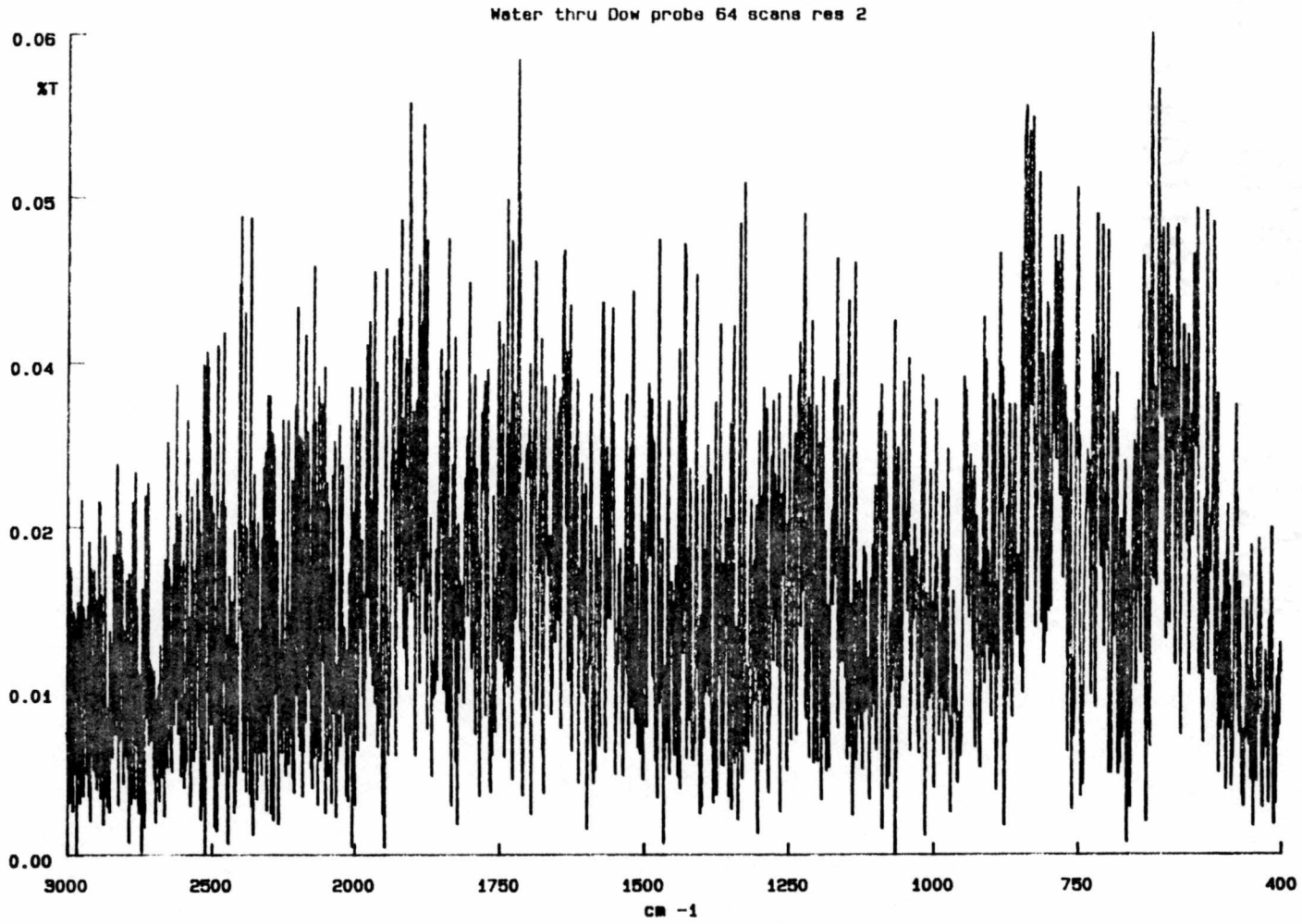


Figure 3-9. Raman Spectrum of Water

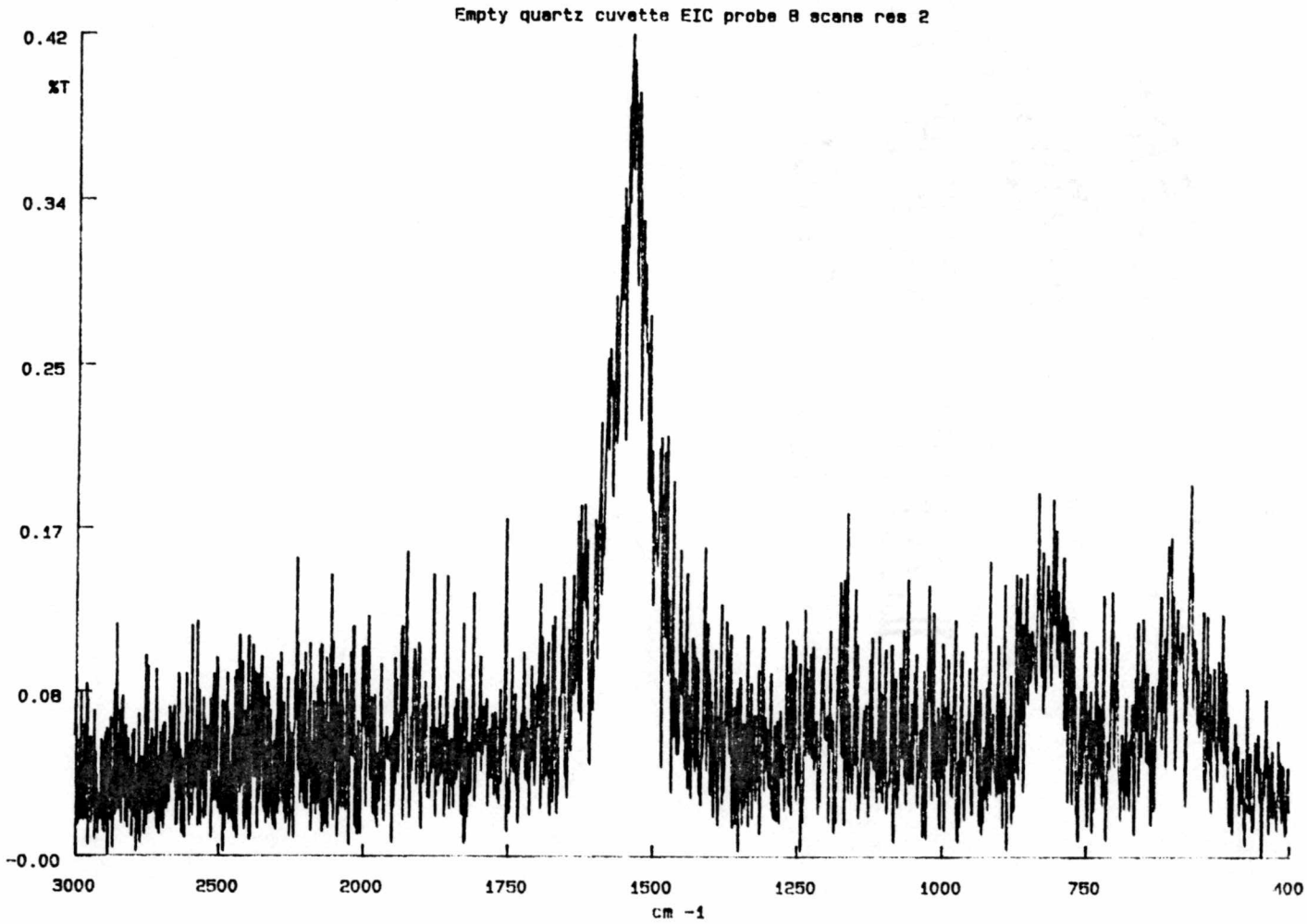


Figure 3-10. Raman Spectrum of Empty Quartz Cuvette

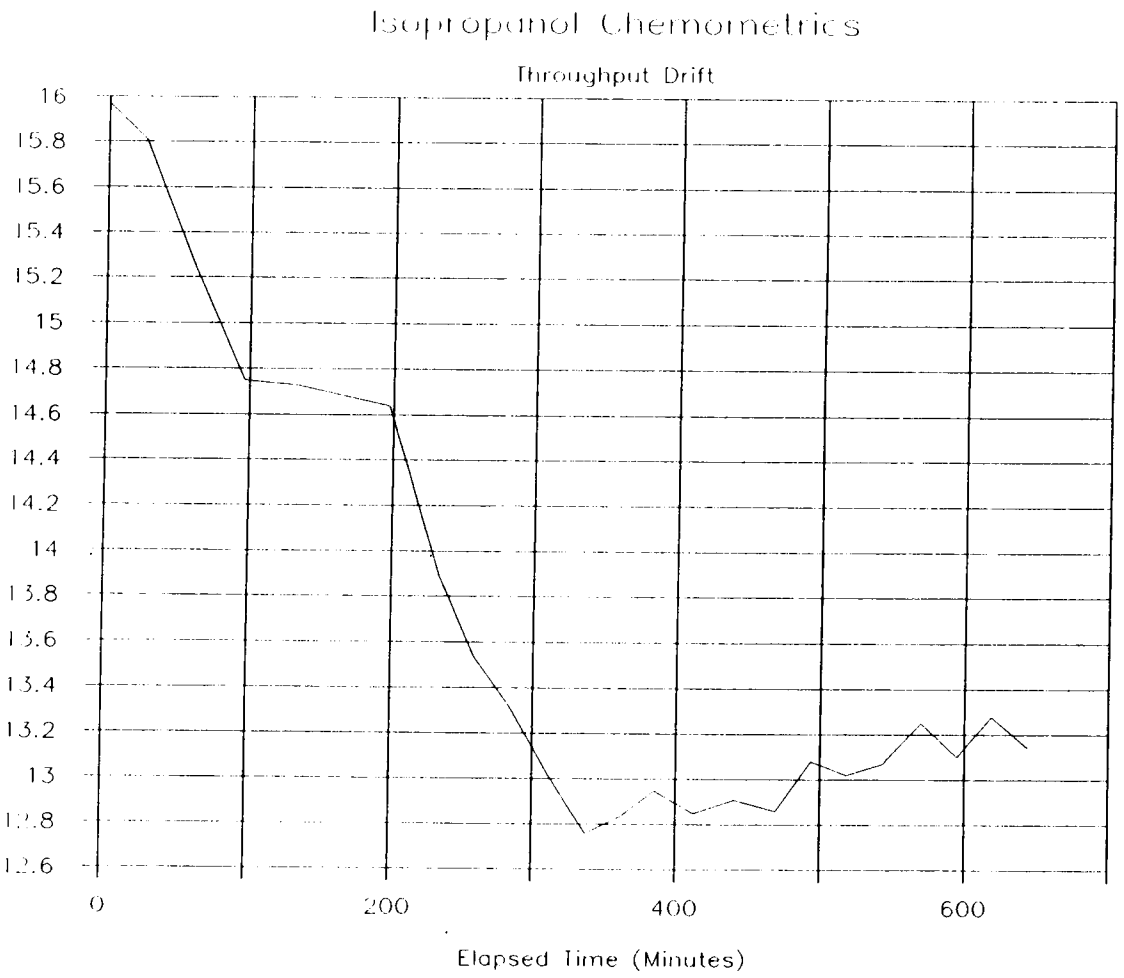


Figure 3-11. Power Drift of Benzene

taken with the same benzene sample every half hour. The laser was left on for the entire period.

These results demonstrate the need for a calibration standard. When a large number of readings is taken for chemometrics, a reading against a known calibration standard should be taken from time to time. The spectra may need to be multiplied by a correction factor if the peak intensity of the calibration standard drifts.

CHAPTER 4

PROPERTIES OF LONG FIBERS

Summary of Discoveries

Since the properties of the long fibers will affect the signal to noise performance of the analyzer, it was necessary to measure them. Two bundles of 400 micron core silica core silica clad fiber were tested. The bundles had seven fibers, one for excitation and six for receiving. One bundle was 35 meters long, the other was 10 meters long.

The following properties were observed. Numerical aperture was approximately 0.33. Connector loss per connection was 0.9 dB average, with a standard deviation of 0.16 dB. Fiber attenuation was too small to measure accurately. For consistently adequate signal to noise performance for the Eastman mixture through long fibers, a laser that delivers 2 watts to the microscope objective is needed. Fiber performance was independent of temperature in the 0°C to 40°C range. Significant silica Raman scattering occurred in both the 10 and 35 meter fibers.

Numerical Aperture and Attenuation

Numerical aperture was measured by the method described by Newport and found to be about 0.33. [Newport 86] See Appendix F.

Standard SMA connectors and bushings are used to connect the fiber bundle to the fiber optic probe and the spectrometer head. As reported in Appendix G, for 108 measurements of several connector combinations, the loss per connection was found to have a mean of 0.9 dB and a standard deviation of 0.16 dB.

It was expected that an estimate of the attenuation of the long fibers in the Raman wavelengths could be obtained by comparing the intensities of spectra taken through fibers of different lengths while keeping the laser power and other factors constant. However, it soon became apparent that the attenuation in the fibers at Raman wavelengths was much smaller than the uncertainty in the other steps of the measuring procedure. For example, consider the intensities of acetic acid spectra listed in Table 4-1.

Table 4-1. Intensities of Acetic Acid Spectra

Spectrum	Fiber Length (m)	Scans	Acetic Peak
a162235	35	16	0.49
a16220	0	16	0.44
a1282235	35	128	0.43
a128220	0	128	0.42

(Note: Fiber length of 0 means that the connector at the end of the 1 meter probe pigtail was placed directly into the focus of the 5X microscope objective. The 35 meter fiber was connected to the probe through a bushing.)

Remarkably, the power intensity of the spectrum is a little higher for the 35 m fiber and one splice than coupling directly into the probe excitation fiber pigtail. This indicates that the amount of attenuation in the fiber is less than the amount of fluctuation in the laser power and the reproducibility in the laser to fiber coupling.

An experiment to obtain an estimate of attenuation at excitation wavelength was conducted as follows. The laser output was coupled to a 10 meter excitation fiber through a 5X objective, and power into the objective and out of the far end of the fiber were measured. The same procedure was

repeated for the 35 meter excitation fiber. It was found that the 35 meter bundle output was .02228 dB less than the output of the 10 meter bundle. Since all other conditions were the same, the .02228 db in added loss is attributed to the 25 meters of added length in the longer fiber. Multiply this value by 40 and an estimate of 0.89 dB/km is obtained. For 70 meters of fiber, the attenuation is .07 x .89 = 0.062 dB. The fiber loss is less than the 0.16 dB standard deviation in connector loss described earlier.

Signal to Noise Performance

Two bundles of 400 micron core fibers were tested. Each bundle had seven fibers, one for excitation, and six for receiving. One bundle was 35 meters long, and the other was 10 meters long. Several spectral observations were made using the 35 m bundle. Laser power on input side of 5X objective was measured to be 440 mw. Raman spectra for 25 ml of each sample were taken. The desired signal is the peak value of the Raman feature of the sample. The undesired signal is the peak of the silica Raman feature. Noise is the peak in the 2520-1820 cm-1 band. Receive fiber was 35 meters for all observations. Results are listed in Table 4-2.

Table 4-2. Signal to Noise Ratios

Sample	Scans	Rate	Excitation Fiber Length	Desired Signal	Undesired Signal	Noise	Signal to Noise
benzene	16	2	35	6.84	0.97	0.11	62.18
benzene	96	2	35	7.04	0.94	0.06	117.33
benzene	8	1	35	7.21	0.97	0.12	60.08
benzene	48	1	35	7.35	0.95	0.05	147.00
cumene	16	2	35	1.93	0.87	0.11	17.55
cumene	96	2	35	1.86	0.78	0.05	37.20
cumene	8	1	35	1.82	0.76	0.09	20.22
cumene	48	1	35	1.85	0.77	0.05	37.00
isopropanol	16	2	35	0.83	1.16	0.11	7.55
isopropanol	96	2	35	0.80	1.18	0.05	16.00
isopropanol	8	1	35	0.83	1.23	0.12	6.92
isopropanol	48	1	35	0.80	1.18	0.06	13.33
acetic	16	2	35	0.49	1.35	0.10	4.90
acetic	128	2	35	0.43	1.28	0.05	8.60
acetic	8	1	35	0.44	1.34	0.09	4.89
acetic	64	1	35	0.44	1.29	0.04	11.00
acetic	16	2	0	0.44	0.56	0.09	4.89
acetic	128	2	0	0.42	0.51	0.04	10.50
acetic	8	1	0	0.45	0.46	0.10	4.50
acetic	64	1	0	0.43	0.45	0.05	8.60

One conclusion is apparent from the data. For acetic acid spectra with 17.3 minute collection times, the signal to noise ratio varies from 8.6 to 11. A signal to noise ratio of 11 was obtained for the acetic acid spectrum taken during the short fiber chemometric experiment (Chapter 2). To obtain dependable chemometric quality spectra with the long fiber system in 2.2 minutes, the performance of the present system must be improved by a factor of 2.83 to reduce the collection time, and by another factor of 11/8.6= 1.28 for consistent performance. The total improvement needs to be 2.83 x 1.28 = 3.62 = 5.6 dB over the current system, which uses 440 mw input to the microscope objective. One way to do this would be to use a two watt rated laser that actually delivers two watts to the objective.

Temperature Sensitivity of Long Fiber

White light response of the 35 meter fiber bundle was measured by shining a flashlight into the fiber bundle and taking the spectrum with the interferometer. The flashlight spectrum was also measured directly with the interferometer. The fiber spectrum was divided point by point by the interferometer input spectrum to yield wavelength response of the fiber. Responses were collected for the fiber at 23°C and at 40°C. Both spectra are essentially flat throughout

the 2940 through 580 wave number band, with the exception of a dip in the wavelength response in the 2289-2089 wave number band. The wavelength responses are shown in Figures 4-1 and 4-2. The dip is 5.66 dB down from peak at 40°C and 5.42 dB down from peak at 23°C. This temperature change causes no significant change in the wavelength response of the fiber.

Cyclohexane spectra were also obtained with 2.2 minute collection times at various temperatures to determine if the collected spectrum changed as a function of fiber temperature. Spectra were collected utilizing 35 meter fibers on both the excitation and receiving sides. At room temperature with 230 mw at the sample, the maximum band height was 1.18. After the fibers were cooled in ice water for an hour, the signal level decreased slightly to 1.12. On a different day, a new spectrum was taken at room temperature with a power level of 340 mw at the sample (Figure 4-3). The maximum band height was 1.90. The fibers were warmed in water to 40°C for an hour, and the maximum band height was 1.89. Comparison of the spectra shows that no new features were introduced due to changing temperatures.

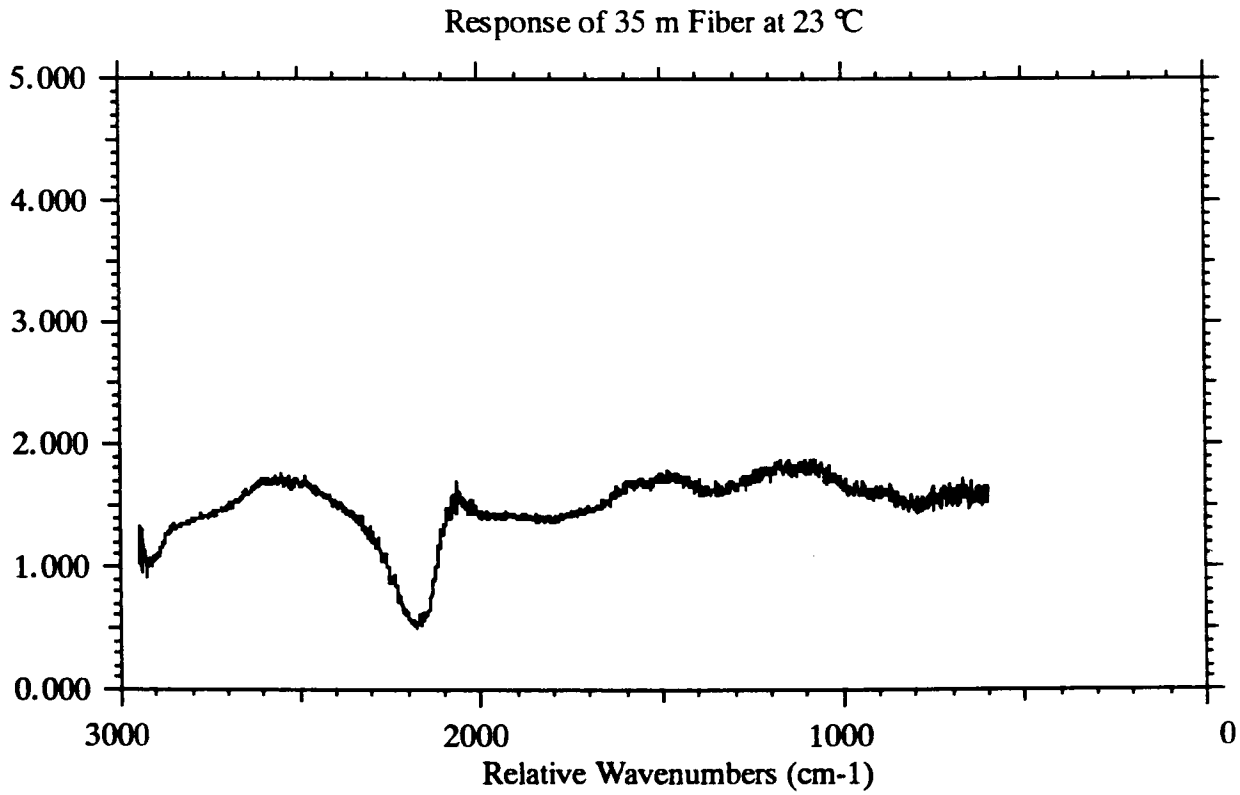


Figure 4-1. Response of 35 m Fiber at 23°C

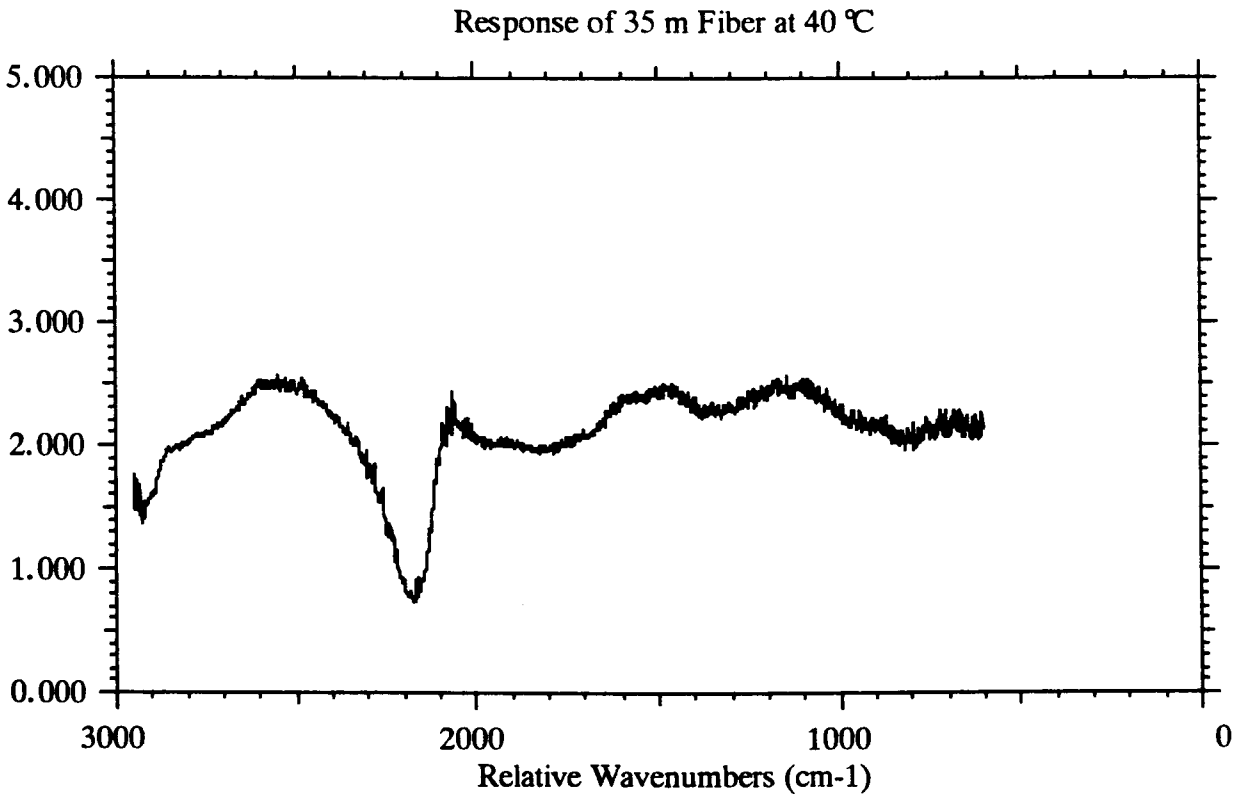


Figure 4-2. Response of 35 m Fiber at 40°C

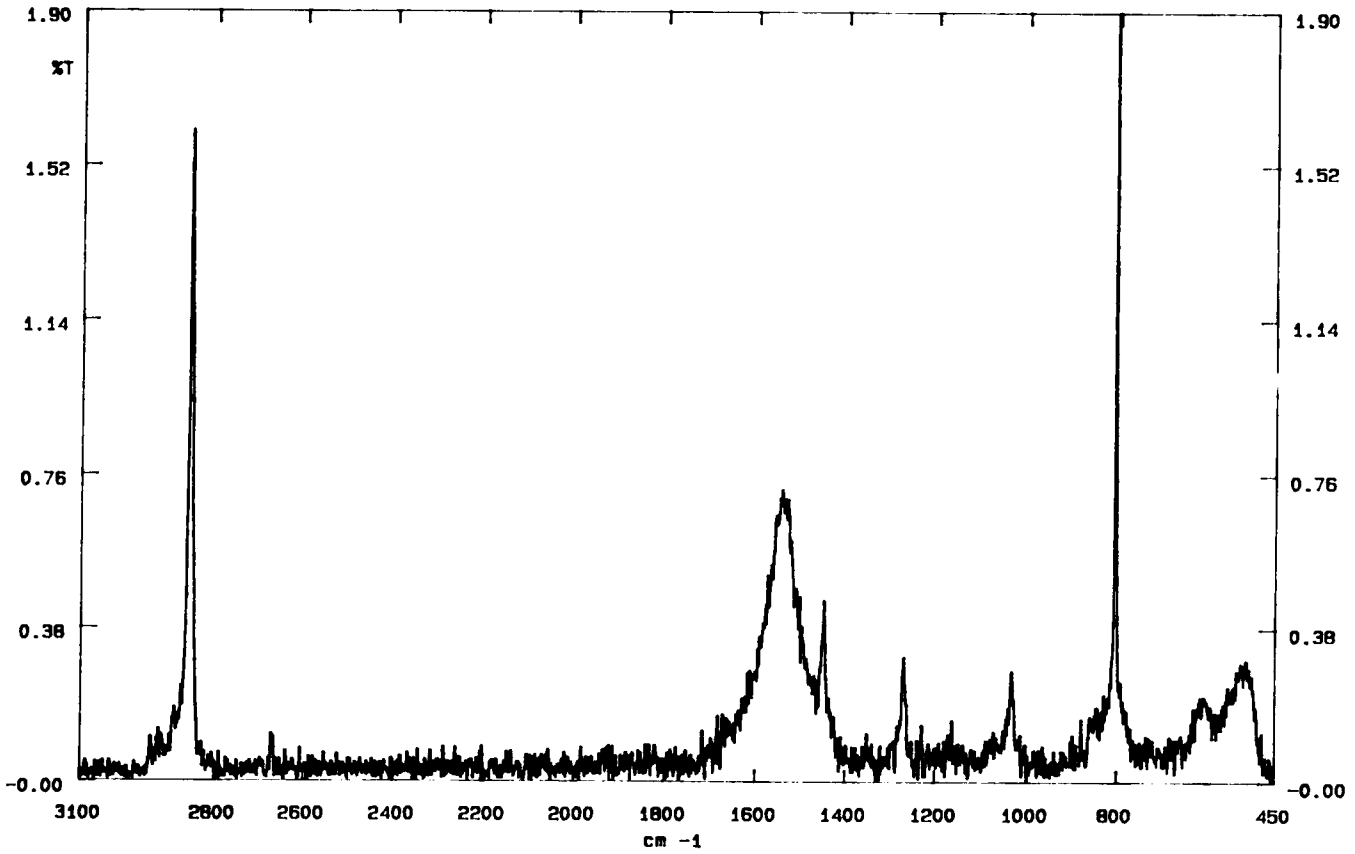


Figure 4-3. Cyclohexane through Fiber at 23°C

The conclusion of these experiments is that temperature variation over the range of 0°C to 40°C does not affect the performance of the 35 meter fiber bundle.

Silica Features in Long Fibers

All the spectra taken through the long fibers include broad Raman features with local peaks at 1535, 817 and 516 wave numbers. These can be seen in the 35 meter fiber spectra in Appendix C. The intensity of the peaks is found to vary with the length of the fibers. Other researchers have attributed these bands to Raman scattering due to residual hydroxyl in the silica fiber core. [Williams 90]

The silica Raman feature at 817 wave numbers is broad enough to cause conflicts with many chemicals of interest. These include the acetic acid feature at 893 cm-1, the solvent feature at 821 cm-1, and the isopropanol feature at 825 cm-1. These are the features that produced the short fiber chemometric results.

One possible way of overcoming errors introduced into the chemometric predictions due to this silica feature is to perform chemometric analyses based on spectral features at

larger Raman shifts than the silica features. No major silica features were observed at Raman shifts greater than 1750 cm-1. With liquid nitrogen cooling, the detector half power offset is 2892 cm-1 for white light. At room temperature, the detector half power offset is 3422 wave numbers. By using a cooling temperature higher than liquid nitrogen, the acetic feature at 2947.5 cm-1 becomes very strong, and the isopropanol/solvent feature at 2875 cm-1 would be strengthened. This may provide better chemometric predictions than the features in the 900-800 cm-1 band. The higher noise due to higher detector temperature might require some additional laser power. This approach would eliminate the need for liquid nitrogen.

Since the cumene oxidate components are stronger Raman scatterers than the components of the other mixtures, and since they fall between the silica features, there should not be much of a problem with the cumene predictions. The cumene feature at 746 cm-1, the acetophenone feature at 732 cm-1, the dimethyl benzyl alcohol feature at 725 cm-1, and the cumene hydroperoxide feature at 718 cm-1 fall between the 817 and 516 cm-1 silica features.

To test the theory that the silica features might be amplified by reflections off the inside of a small vessel, acetic acid spectra were taken with the probe at the 25 ml

mark in a graduated cylinder, and with the probe in a 1 liter flask. There is virtually no difference in the two spectra. This suggests that the silica features are not amplified by internal reflections in the small vessel.

Linearity of Raman Features

To test the linearity of Raman features, acetic acid spectra were observed at different excitation levels. The acetic acid peak (893 cm-1) and the 1550 cm-1 silica peak fit closely to straight lines determined by least squares. [Scheid 68, pp. 235-236] See Table 4-3 and Figure 4-4.

Table 4-3. Height Versus Excitation

Excitation (mw)	Acetic Intensity (Least Squares)	Silica Intensity (Least Squares)	Acetic Intensity (Actual)	Silica Intensity (Actual)
320	0.27833333	0.8925	0.27	0.88
280	0.24380952	0.7775	0.27	0.80
240	0.20928571	0.6625	0.21	0.67
200	0.17476190	0.5475	0.15	0.53
160	0.14023809	0.4325	0.14	0.44
120	0.10571428	0.3175	0.10	0.30
80	0.07119047	0.2025	0.08	0.20
40	0.03666666	0.0875	0.04	0.10

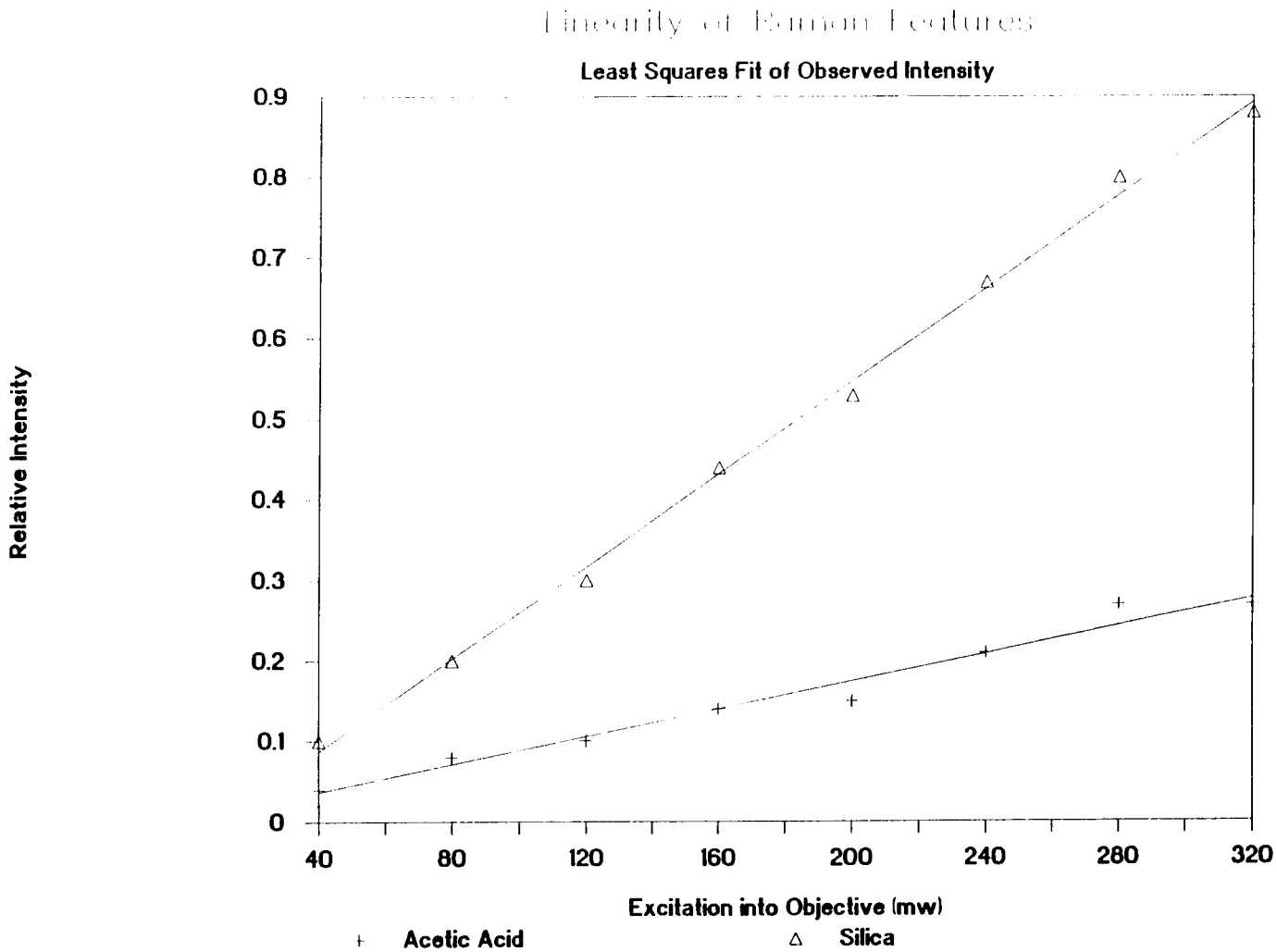


Figure 4-4. Linearity of Raman Spectra

Silica Intensity as a Function of Fiber Length

The silica features of the long fibers are quite noticeable. Unfortunately, they occur in the same bands as the most useful acetic, solvent and isopropanol features.

Comparative spectra of acetic acid were taken through various combinations of fibers to identify the mechanism(s) generating the silica features. See Table 4-4 and Figure 4-5.

Table 4-4. Silica Raman Intensity

Fiber Length	1550 cm-1 Intensity
Excitation 35 meters Receive 35 meters	0.86
Excitation 35 meters Receive 0 meters	0.75
Excitation 0 meters Receive 35 meters	0.26
Excitation 0 meters Receive 0 meters	0.08
Excitation 10 meters Receive 10 meters	0.34
Excitation 10 meters Receive 0 meters	0.29
Excitation 0 meters Receive 10 meters	0.19

Even without a fiber bundle coupled in, there is a silica feature of 0.08. The noise in these spectra is about 0.04. Thus, even at this low level, there is some silica signal above the noise. This is apparently contributed by the probe itself.

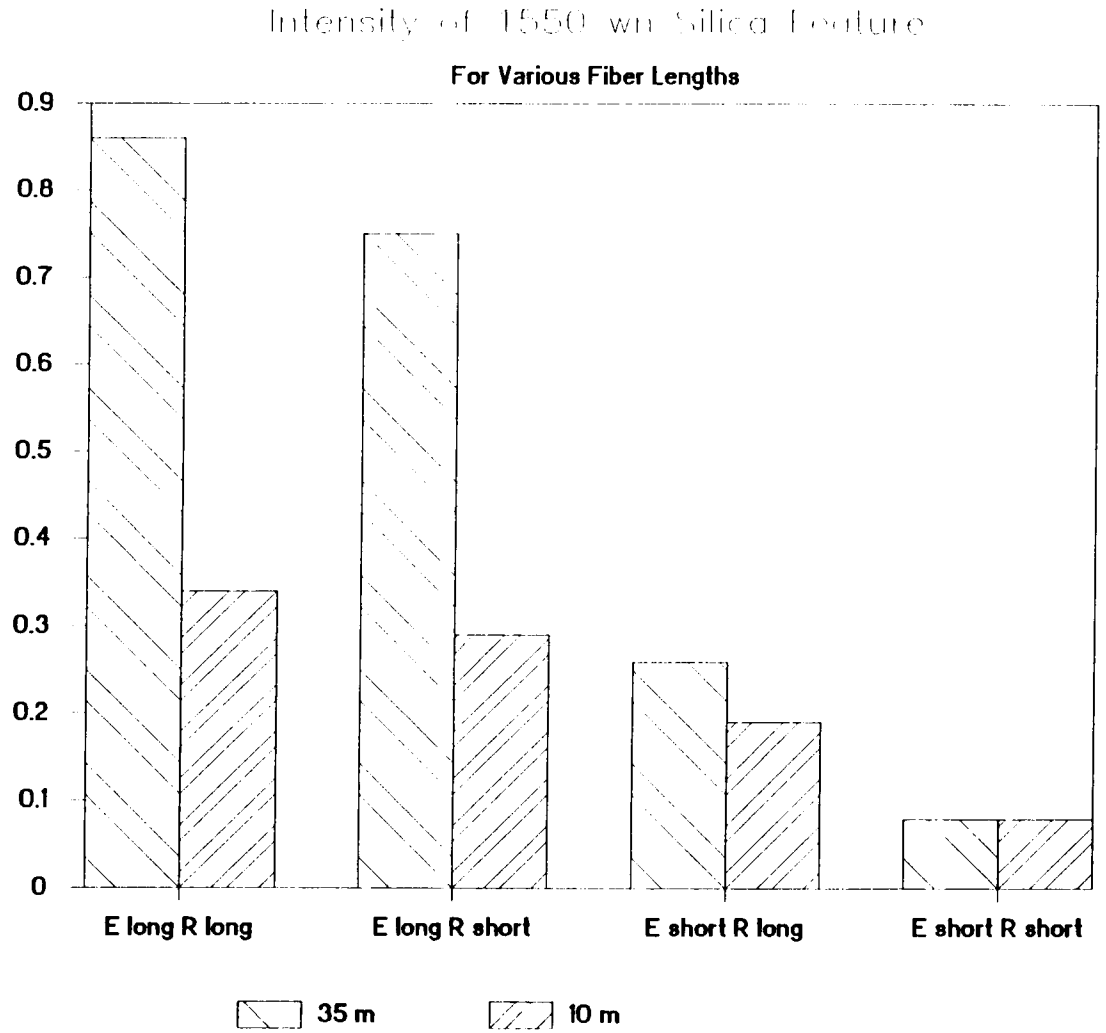


Figure 4-5. Silica Raman Line Height

The signal with a long excitation fiber and no added receiving fiber is almost as strong as the signal with both long excitation and receiving fibers. In both cases, the Raman signal strength seem to increase linearly with the length of the fibers. This suggests that placing a filter at the connection between the excitation fiber in the bundle and the excitation fiber in the probe should greatly reduce the silica feature. Other researchers have successfully used filters to suppress Raman energy generated in the fiber. [Myrick 90b]

The feature contributed by the receiving fiber is much smaller than the feature contributed by the excitation fiber. The two components do not add algebraically to give the entire silica feature. The feature produced by the receiving fiber does not appear to increase as rapidly with length as the transmit feature. Therefore, efforts to filter the Rayleigh line out at the probe connector do not appear cost effective.

The conclusion of these observations is that Rayleigh scattering by the sample, of Raman features generated in the excitation fiber is the strongest contributor to the silica features. A small contribution is added by Raman scattering in the receive fiber, of the Rayleigh energy generated in the sample. Dramatic reduction of the silica features should

be realized by using a filter at the probe excitation pigtail.

Overcoming the Silica Raman Problem

There are several ways of dealing with the silica Raman problem. The simplest is to ignore it if possible. Although the Eastman mixture shows very strong silica Raman features for spectra taken through 35 meter fibers, the chemometric errors were within acceptable limits. Unfortunately, this will probably not work if the fibers are on the order of hundreds of meters long. It is likely that the silica features would swamp the sample features.

Another approach would be to use a thermoelectrically cooled detector instead of liquid nitrogen cooling. The higher temperature would increase the wavelength response of the detector, and enable chemometrics to be carried out in the high Raman shift region where sample features are present, but Raman features due to the silica are not significant.

One way of greatly reducing, but not eliminating the problem, is to insert a line filter at the connector between the long excitation fiber and the probe excitation fiber pigtail. This would eliminate the strong contribution of the excitation fiber to the problem. Graded index (GRIN) lenses

could be used to couple the fiber ends to the filter. Due to its high insertion loss (at least 55 per cent), the line filter should be removed from the laser if this configuration is used. [Melles Griot 85, p. 281]

The scheme with GRIN lenses and a filter is a variant on the dual fiber, dual filter, optrode scheme developed by Myrick and Angel. [Myrick 90b] In this approach, two optical fibers are used, one each for excitation and receiving. Each fiber has a filter mounted directly on the end. The two fiber ends are aligned, with the sample between them. The authors report that this configuration gives much stronger Raman pickup than an angled probe. By using a long pass filter on the receiving fiber, the laser energy at excitation wavelength is kept out of the receive fiber. One problem with filters is that the filter material fluoresces, and the effect must be subtracted out of the spectrum.

CHAPTER 5

COUPLING A LASER TO A FIBER

Principles of Coupling

The object of coupling is to efficiently couple the energy from a 1.064 micron wavelength laser beam with a Gaussian intensity distribution into a 400 micron core diameter fiber. Coupling is done by using an optical element such as a lens or a microscope objective to capture the laser light and focus it onto the fiber core. For optimal coupling, the beam should be focused down to a spot comparable to the core size and centered on the center of the fiber core. The angle of the outermost rays being coupled into the fiber must not exceed the angle of acceptance of the fiber. [Newport 87, p. J-3]

Staying within the angle of acceptance is not difficult. The numerical apertures for various fibers have been measured and found to be either 0.22 or 0.33. (See Appendix F) For conservative design, a value of 0.22 is assumed. A coupling element with a numerical aperture of less than 0.22 should be suitable.

It is easy to focus the beam to a suitable spot size. The diffraction limited waste size is given by

$$w = (4\lambda f)/(\pi D)$$

where λ is the wavelength, f is the focal length of the coupling element, and D is the diameter of the beam incident on the coupler. [Newport 87, p. J-4] The beam diameter just in front of the coupler has been measured and found to be 3.5 mm. The focal length of a Newport 5X objective is 25.5 mm. [Newport 87, p. H-24] The beam wavelength is 1.064 microns. These result in a waist diameter of 9.87 microns, considerably smaller than the 400 micron diameter of the fiber core.

Reflections and Anti-Reflection Coatings

For a normally incident light beam on an air-glass interface, 96 percent is transmitted and 4 percent is reflected. [Melles Griot 85, p. 71] Thus, there is a reflection loss of 4 percent (0.177 dB) per face in an optical system with multiple elements. This acts to limit the efficiency of a coupler with several air to glass interfaces.

This can be countered with an anti-reflection coating. A simple anti-reflection coating is a quarter wavelength thick at the center of the passband, and has an index of refraction that is the square root of the products of the indices of refraction of the two media. The behavior of an anti-reflection coating is analogous to that of a quarter wave impedance matching transformer in a transmission line. More sophisticated anti-reflection coatings are made up of multiple dielectric layers. At the center of the passband, an anti-reflection coated lens may have a transmission ratio in excess of 99.75%. [Melles Griot 85, p. 78] Newport microscope objectives are anti-reflection coated, but the coating is optimized for visible light. The Newport objective has transmission of 97.5% per face at 1.064 microns. [Newport 89]

Reflection Limits on Coupling Efficiency

There is a limiting value of coupling efficiency which is dictated by reflection losses. In the configuration used in this experiment, there is an air gap between the coupling device and the excited end of the fiber, and another air gap between the output end of the fiber and the power meter sensor. Loss due to reflection at the fiber ends is figured thus:

Transmission per face:	0.96	=	-0.177 dB
Transmission through two faces:	0.9216	=	-0.355 dB

The Newport 5X objective consists of a single lens. Reflection losses of the 5X objective are figured thus:

Transmission per face:	0.975	=	-0.110 dB
Transmission through two faces:	0.950625	=	-0.220 dB

The 10X and 20X objectives have two lenses or four faces. Reflection losses for two lens objectives are figured as follows:

Transmission per face:	0.975	=	-0.110 dB
Transmission through four faces:	0.903687	=	-0.440 dB

For an uncoated single lens, the transmission per face is 96%. Each lens has two faces. The reflection loss for an uncoated lens is figured thus:

Transmission per face:	0.96	=	-0.177 dB
Transmission through two faces:	0.9216	=	-0.355 dB

Although it was not used in this experiment, Newport recommends using an FL-10B diode laser objective for this

application. Its transmission is 94% (-0.269 dB) total for the device at 1.064 microns.

To find the minimum possible value of loss for each system, add the dB losses due to the device to the dB reflection losses of the fiber as in Table 5-1.

Table 5-1. Reflection Limited Minimum Losses

	5X Obj	10X Obj	20X Obj	Uncoated Lens	FL-10B
Device (dB)	-0.220	-0.440	-0.440	-0.355	-0.269
Fiber (dB)	-0.355	-0.355	-0.355	-0.355	-0.355
Total (dB)	-0.574	-0.794	-0.794	-0.709	-0.623
Total (%)	87.6%	83.3%	83.3%	84.9%	86.6%

The values in Table 5-1 are the reflection limited values of coupler transmission efficiency ignoring loss mechanisms in the fiber, and assuming that alignment and all other considerations are perfect.

Experimental Procedure

The output of the CVI C95 1.064 micron laser was coupled into the excitation fiber (400 micron core diameter) of the Dow 6 around 1 probe with various coupling devices. The position of the excitation end of the fiber with respect to the coupler focus was adjusted with a three axis positioner for maximum output at the probe end.

The output power was measured at the probe end through a 1 mm quartz window with a Coherent 210 optical power meter. Input power going into the coupling device was also measured with the power meter.

In addition, the pattern of the infrared light emerging from the probe end was observed by projecting it onto a target consisting of concentric circles spaced 1 mm apart, and observing the resulting pattern through an Electro-physics 6100M Infrared Viewer. The diameter of the infrared beam incident on the coupling device was measured by the same method.

Observed Data

The observed efficiency of coupling the laser into the excitation fiber of the Dow probe with various devices is shown in Table 5-2.

Table 5-2. Observed Coupling Efficiencies into Dow Probe

Objective	Input (mw)	Output (mw)	Observed Efficiency (%)	Observed Efficiency (dB)	Reflection Limited Efficiency (%)
5X	640	470	73.4%	-1.341	87.6%
10X	650	440	67.7%	-1.695	83.3%
20X	660	390	59.1%	-2.285	83.3%

The performances indicated in Table 5-2 are much lower than the reflection limited maxima. The Dow probe has been used for many experiments, and its performance appears to have deteriorated with use.

Note that in a similar experiment in which the probe was replaced with a one meter fiber with factory installed connectors and the 5X objective was used for coupling, an input to the objective of 650 mw resulted in an output from the fiber of 560 mw, an efficiency of 86.15%. This is very close to the reflection limited value of 87.6%. Also, in previous experiments, efficiencies as high as 83% were noted

using the 5X objective and the Dow probe. Thus, it is possible to obtain couplings very near the reflection limited maximum with the 5X objective.

To attain high efficiency, it is necessary to both align the objective so that the laser beam is hitting it exactly on center, and also to align the fiber end so that it is precisely at the focus of the objective.

The patterns of the output of the probe for various coupling devices were also observed with the infrared viewer and the calibrated target. They are described in Table 5-3.

Table 5-3. Fiber Output Radiation Patterns

Device	Pattern
Objectives:	
5X	Circular spot subtending an angle of .161 radians
10X	Circular spot subtending an angle of .207 radians
20X	Concentric rings subtending the following angles: Inner ring subtends from .12 to .16 radians Outer ring subtends from .2 to .44 radians
Lenses:	
50 mm	Single ring subtends from .27 to .35 radians
125 mm	Circular spot subtending an angle of .162 radians
150 mm	Circular spot subtending an angle of .162 radians
225 mm	Circular spot subtending an angle of .147 radians

In addition, with the infrared viewer and the calibrated target, the beam incident on the coupling device was observed to be circular with a diameter of 3.5 mm.

Conclusions

Both in terms of reflection limited maximum efficiency and observed efficiency, the 5X objective is the most efficient coupler. With reasonable care, it is possible to couple to within 0.07 dB of the reflection limited efficiency.

Objectives whose numerical aperture is greater than the numerical aperture of the fiber do not couple as efficiently as objectives whose numerical aperture is less than the numerical aperture of the fiber.

Since a graded index (GRIN) lens is only 2 mm in diameter, it is not possible to use it to efficiently couple a 3.5 mm diameter beam into the fiber.

CHAPTER 6

THE FIBER OPTIC PROBE AND SAMPLE CELL

Mathematical Model of the Probe

A model of the probe operation has been developed by Dr. Michael Roberts based on the work of Plaza and others. [Garrison 89, Appendix A] [Plaza 86] Consider two fibers whose ends have been polished perpendicular to their optical axes, and whose optical axes intersect as shown in Figure 6-1. The coordinate system is arranged such that the origin is at the center of the polished face of the excitation fiber. The z-axis is the optical axis of the excitation fiber. The y-axis is in the plane formed by the optical axes. The x-axis is normal to the plane formed by the optical axes.

The ends of the fibers are displaced and rotated with respect to each other. The optical axes intersect at an angle α. The two nearest corners of the fibers are separated by a displacement, d_y parallel to the y-axis and d_z parallel to the z-axis. The excitation fiber emits light in a cone. The angle of the cone is θ_{ce}, the critical angle with respect to the excitation optical axis. The receiving fiber accepts light that is incident within a cone of

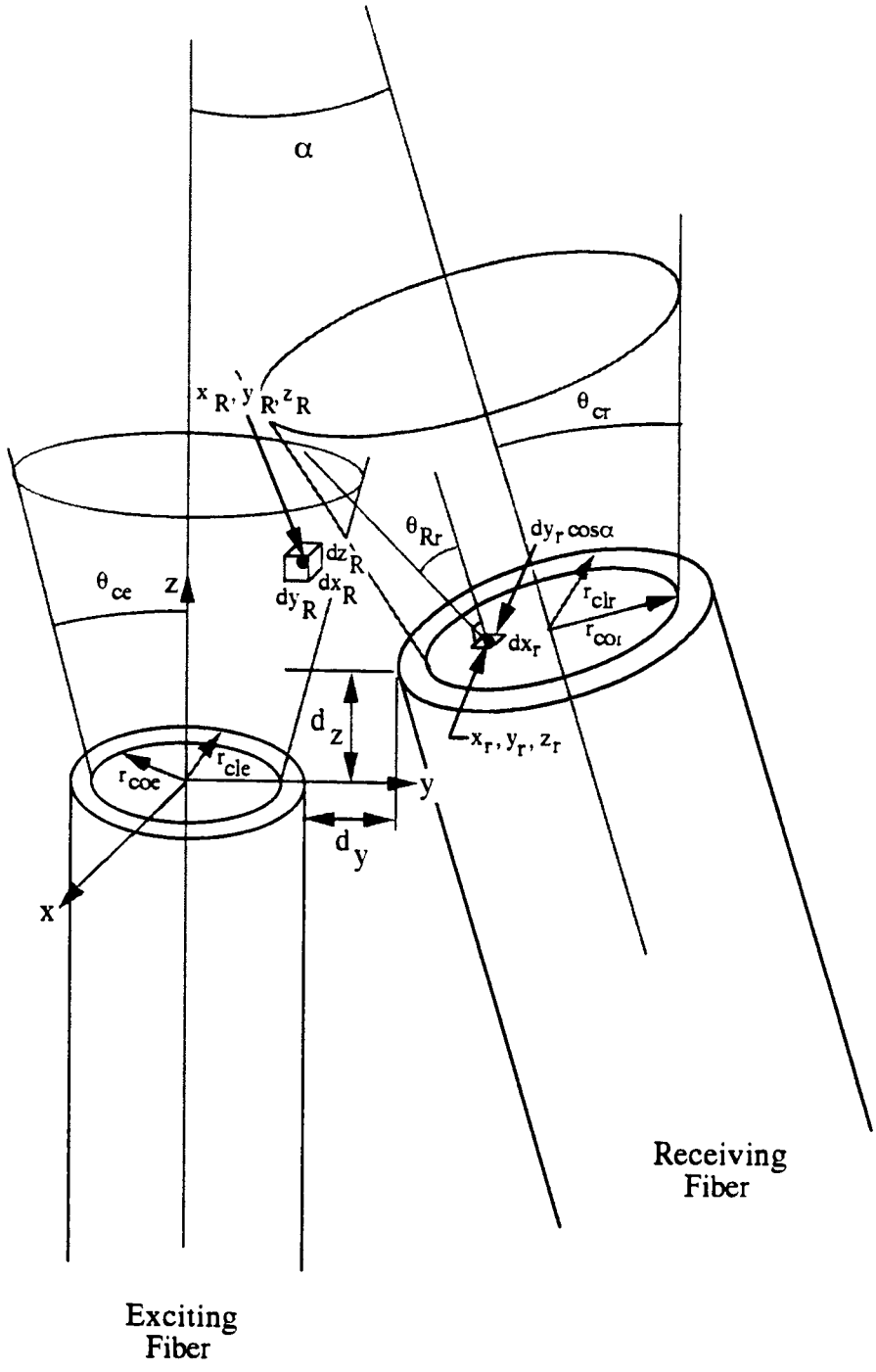


Figure 6-1. Probe Coupling

acceptance. The angle of the cone is θ_{cr}, the critical angle with respect to the receiving optical axis. Some simplifying assumptions are made about the excitation power distribution. The power from the excitation fiber is assumed to be uniformly distributed through the circular cross section of the cone. Thus, the local power density is the total excitation power P_0 divided by the area of the circle. Attenuation of the medium is assumed to cause exponential decay of the power density as a function of displacement along the z-axis. That is, the local power density is multiplied by a factor of $e^{-z/z0}$, where z0 is the space constant.

The scattering mechanism is Raman scattering. The amount of Raman energy generated at a point is proportional to the local power density. It is also proportional to the local density of scatterers, N, (number of scatterers per unit volume). Not all the power incident on a point is Raman scattered. The proportion scattered is given by β, the Raman scattering cross section, a property of the scattering medium. Within a given incremental volume, there are a large number of scatterers, each scattering its Raman photons in random directions. Due to this randomness, it is assumed that the Raman power radiated by a given point is uniformly distributed over a sphere (or over a solid angle of 4π steradians) centered on the point.

The coupling to the receiving fiber is based on the cone of acceptance. If the Raman power originating at a point in space is within the receiving cone of acceptance, and if the ray from the point in space to a point on the receiving face is within the receiving critical angle, then the power is accepted by the receiving fiber. Otherwise, it is rejected.

The amount of Raman power coupled into the receiving fiber is based on simple geometry. Since the Raman power emanating from a point in space is assumed to be uniform over 4π steradians of solid angle centered on the point, it is assumed that the proportion of the power coupled into an incremental area on the receiving fiber face is the proportion of solid angle subtended by the incremental area. In other words, the amount of solid angle in steradians centered on a point and subtended by the incremental area divided by 4π steradians, gives the proportion of Raman power originating at the point that is coupled into the incremental area on the receiving face.

Given these assumptions, a formula for the total Raman power coupled into the receiving fiber can be derived. For an incremental area on the receiving surface, the total power received is the sum of the powers received from each

of the incremental volumes in the excitation cone. The power generated by each incremental volume is calculated as described above; the summation occurs through the three dimensional space of the excitation cone. The total power coupled into the receiving fiber is the sum of all the powers coupled into each incremental area on its face; the summation occurs over the two dimensional area of the receiving face. The result of the summations is a five dimensional integration :

$$P_r = \frac{P_0 \beta N}{\pi} \int_{\substack{\text{receiving fiber} \\ \text{end surface}}} \left\{ \int_{\substack{\text{exciting fiber} \\ \text{cone volume}}} \frac{a(z_R) r_r r_R (d_{rR})_{z_r} dr_r d\theta_r dr_R d\theta_R dz_R}{[r(z_R)]^2 \left[(d_{rR})_{x_r}^2 + (d_{rR})_{y_r}^2 + (d_{rR})_{z_r}^2 \right]^{\frac{3}{2}}} \right\}$$

The excitation cone is symmetrical about the excitation fiber optical axis. If the excitation and receiving optical axes intersect at some fixed angle α, then the coupling is independent of the polar orientation of the receiving fiber. Thus, if another receiving fiber were placed on the opposite side of the excitation fiber, it would receive the same amount of Raman power as the first. Many receiving fibers can be arranged symmetrically around the excitation fiber, and the effectiveness of the probe will be multiplied accordingly.

Computer Implementation of the Probe Model

The integral that models the probe cannot be evaluated in closed form. It must be evaluated numerically. A computer program to perform the integration is listed in Appendix H. The program is written in ANSI Standard C and is portable. It was developed on the VAX, but has also been run on the IBM-PC, the IBM mainframe, and the Cray. Results obtained with the computer program on the VAX are tabulated in Table 6-1.

The precision of the calculations is controlled by the number of increments into which the excitation cone and receiving face are divided. The dividing is done in cylindrical coordinates. In the radial dimension, the space is divided into RADIAL increments. In the azimuthal dimension, the space is divided into AZIMUTH increments. In the axial dimension, the space is divided into AXIAL increments. The receiving fiber face and each axial layer of the excitation cone are divided into the same number of increments. RADIAL, AZIMUTH, and AXIAL are declared constants that are set by the computer user at compile time. The greater the values, the greater the precision, and the greater the cost of the run.

Table 6-1. Computed Coupling Coefficients

Number of intervals in AXIAL dim:		80	80	80	80
Number of intervals in RADIAL dim:		8	4	4	4
Number of intervals in AZIMUTH dim:		16	16	16	16
Exciting core diameter:		200	100	400	200
Exciting clad diameter:		250	125	500	250
Receiving core diameter:		200	200	400	400
Receiving clad diameter:		250	250	500	500
Spacing:		0	0	0	0
Exciting critical angle:		8	8	8	8
Receiving critical angle:		8	8	8	8
Raman cross section:		1	1	1	1
Density:		1	1	1	1
Space constant:		1000000	1000000	1000000	1000000
Analysis depth in core diameters:		450	900	225	450
Axial offset		0	0	0	0
Angle Between Optical Axes:	0	0.001549	0.002161	0.002899	0.003974
	1	0.001716	0.002447	0.003297	0.004497
	2	0.001865	0.002656	0.003515	0.004881
	3	0.001982	0.002864	0.003713	0.005240
	4	0.002090	0.003086	0.003871	0.005467
	5	0.002177	0.003231	0.004010	0.005695
	6	0.002263	0.003353	0.004077	0.005875
	7	0.002321	0.003493	0.004163	0.005997
	8	0.002367	0.003573	0.004212	0.006108
	9	0.002410	0.003640	0.004212	0.006141
	10	0.002433	0.003723	0.004219	0.006201
	11	0.002439	0.003744	0.004143	0.006188
	12	0.002444	0.003760	0.004165	0.006139
	13	0.002440	0.003806	0.004092	0.006110
	14	0.002420	0.003828	0.004016	0.006030
	15	0.002397	0.003765	0.003918	0.005906
	16	0.002367	0.003770		0.005826
	16.5	0.001975	0.002812		0.005615
	18	0.001770	0.002594		0.005181
	20	0.001684	0.002457		0.004908
	30	0.001385			0.003977
	35	0.001277			
	45	0.001121			
	60	0.000988			
	70	0.000942			
Peak Response		0.002444	0.003828	0.004219	0.006201
Peak Angle		12	14	10	10
dB Gain of Peak Angle vs. 0 Angle		1.98	2.48	1.63	1.93

Other parameters of the system are furnished interactively by the user at run time. Core and cladding diameters, spacing, offset and space constant are specified in microns. Spacing is the d_y dimension. Since coupling drops rapidly with increased spacing, the spacing is almost always specified as 0. Axial offset is the d_z dimension. Angles are specified in degrees. Critical angle is the arc sine of the numerical aperture of the fiber. Since only relative performance of alternative probe designs is sought, the Raman cross section and scatterer density are arbitrarily set to 1 for all cases. Since the distillation mixtures are transparent, the space constant (the amount of space that will attenuate the light energy to e^{-1} of its original value) is a large number. If the angle of intersection of the optical axes is small, then the volume of the intersection of the receiving and exciting cones is infinite. If so, then the analysis must be cut off at some arbitrary finite depth. Repeated experience with the model shows that out to a distance of about 90,000 microns, a small but usable amount of Raman energy is coupled into the receiving face. Thus, the number of core diameters should be selected to result in a depth of 90,000 microns. Less depth understates the coupling coefficient; greater depth wastes computer time.

What angle of intersection of the optical axes gives the peak response? For the examples studied, the peak response occurs when the optical axes intersect at angles ranging from 10 to 14 degrees. The Dow patent application says that the response should peak in the range of 10 to 20 degrees. [McLachlan 86] It is noteworthy that while the computer model agrees with what Dow has claimed, both disagree with the experimental observations reported by Hendra. [Hendra 88] He reports that the peak occurs when the angle is zero. It is not clear at present why this apparent contradiction exists.

As already noted, the model shows that for angles for which the volume of the intersection of the cones is infinite, even at an axial distance of 90,000 microns from the probe tip, a small, but finite, amount of Raman scattering still contributes to the total received. This is consistent with the experimental observation (Appendix E) that the end of the probe must be at least 9 cm from the bottom of the vessel before artifacts in the received Raman spectrum due to the bottom of the vessel are buried in the system noise.

To compare alternative designs of multiple receiving fiber probes, the coupling coefficients in Table 6-1 are multiplied by the number of receiving fibers. The result

includes a maximum value of 0.025314. To make the comparison easier the data in the table are normalized by dividing each datum by 0.025314. The results are indicated in Table 6-2 and Figure 6-2.

Implications of the Analysis

These results indicate that it may be possible to greatly reduce the cost of the composition analyzer by careful attention to the probe. A major problem with the present design (which uses Dow's patented probe) is that the system uses long bundles of seven 400 micron low water fibers. At a cost of about $20 per meter, fiber costs are a major component of overall system cost.

The data in Table 6-2 suggest some ways of reducing the cost. According to the model, the response of a probe with a large receiving fiber and an exciting fiber of half the core diameter of the receiving fiber is about 50% greater than the response of a probe whose exciting fiber is the same size as receiving fiber. Thus, a probe with four 400 micron receiving fibers around one 200 micron exciting fiber should perform just as well as a probe with six 400 micron fibers around one. This would reduce fiber costs by at least one third with no loss in system performance.

Table 6-2. Comparative Predicted Probe Performances

Receiving Core Diameter (microns):		200	200	400	400
Excitation Core Diameter (microns):		200	100	400	100
Number of receiving fibers:		6	4	6	4
Angle between optical axes:	0	0.36714861	0.34147112	0.68712965	0.62795291
	1	0.40673145	0.38666350	0.78146480	0.71059492
	2	0.44204787	0.41968870	0.83313581	0.77127281
	3	0.46977956	0.45255589	0.88006636	0.82800031
	4	0.49537805	0.48763530	0.91751599	0.86386979
	5	0.51599905	0.51054752	0.95046219	0.89989729
	6	0.53638302	0.52982539	0.96634273	0.92834004
	7	0.55013036	0.55194753	0.98672671	0.94761791
	8	0.56103342	0.56458876	0.99834083	0.96515762
	9	0.57122540	0.57517579	0.99834083	0.97037212
	10	0.57667693	0.58829106	1.00000000	0.97985304
	11	0.57809907	0.59160938	0.98198625	0.97779884
	12	0.57928419	0.59413763	0.98720075	0.97005609
	13	0.57833609	0.60140633	0.96989808	0.96547365
	14	0.57359563	0.60488267	0.95188433	0.95283242
	15	0.56814411	0.59492770	0.92865607	0.93323852
	16	0.56103342	0.59571778		0.92059729
	16.5	0.46812040	0.44433910		0.88725606
	18	0.41953069	0.40989175		0.81867741
	20	0.39914671	0.38824365		0.77553922
	30	0.32827684			0.62842695
	35	0.30267835			
	45	0.26570277			
	60	0.23417871			
	70	0.22327565			

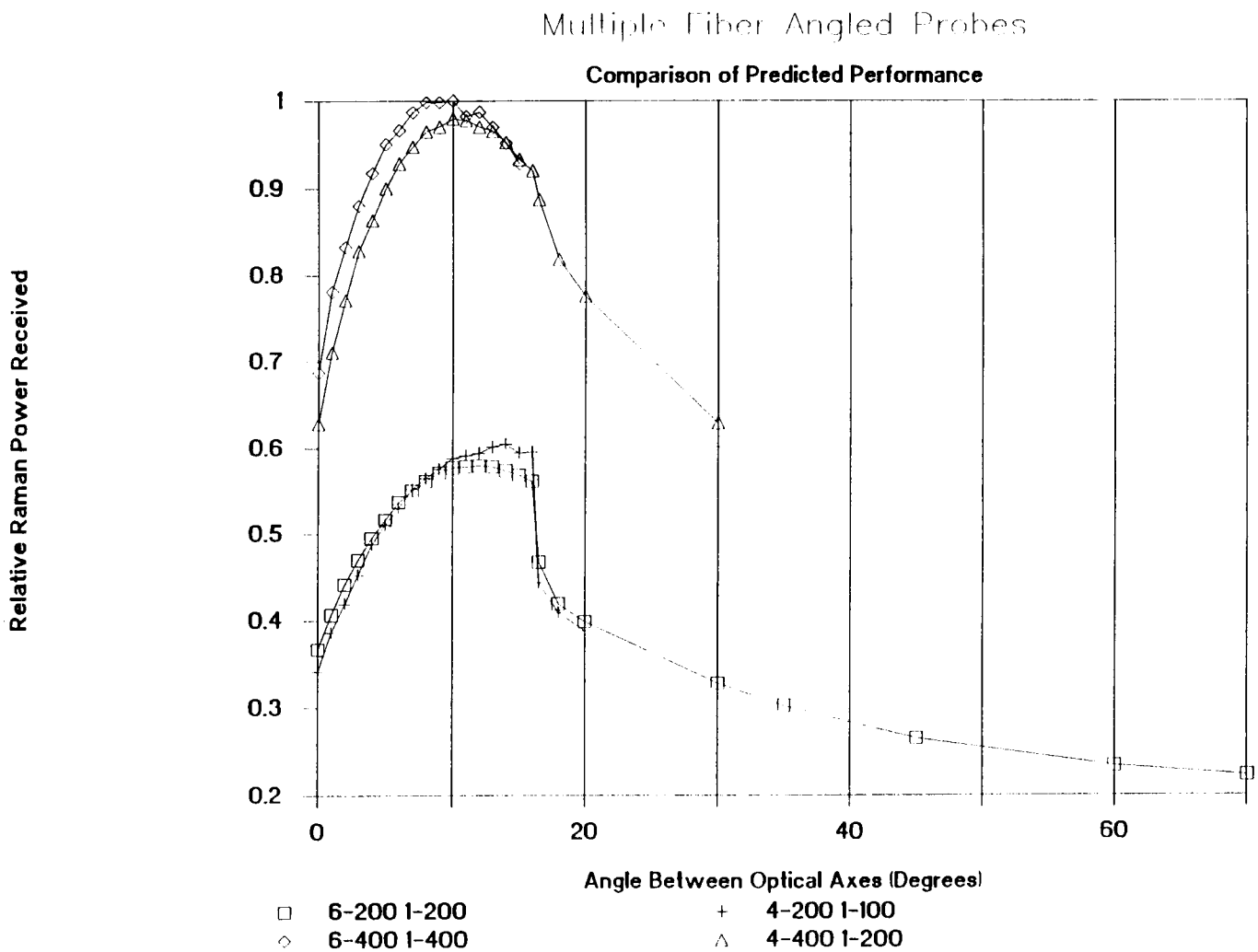


Figure 6-2. Predicted Performance of Four Designs

If some degradation in probe performance is acceptable, even more fiber cost savings are possible. For example, suppose that due to improved detector/amplifier design the signal to noise ratio of the system were raised by three dB at relatively low cost. If all the core diameters in a probe are reduced by half, the overall performance of the probe is reduced by about three dB, but the fiber cost is reduced by at least 75%.

It is unfortunately necessary to mention the computational cost of the present model. High precision is needed to obtain accurate results, but it is somewhat expensive to obtain. A set of 44 data points at 1.3 million calculations per point costs about $332 to generate on the VAX. The time to generate a data point at 160,000 calculations is about 35 minutes on the IBM-AT.

However, the more important problem remains, is the model an accurate description of reality? Dow's reported results, the obvious success of the Dow probe in our system, and the experiment to locate the center of scattering all suggest that it is. Hendra's observations cast doubt. To resolve this question, a research project has been initiated to explore the model further, and to obtain performance versus angle experimental data. The goal is to obtain mathematical and experimental data that are consistent. Once

this is done, it will be possible to confidently base design decisions on the data generated by the model.

An Industrially Hardened Probe

The sample head is the end of the probe that is actually inserted into the sample cell. The sample head is shown in Figure 6-3. [Roberts 90] It consists of a stainless steel body containing seven optical fibers in a close packed bundle, polished on the end, and protected by a quartz window. The central fiber carries the excitation energy to the sample, and the other six collect scattered light from the sample. The collection fibers are angled according to Dow's patented design to maximize collected light.

The probe must be protected from the sample. Early testing of fiber optic probes resulted in damage to the epoxy cement which holds the fibers in place. The epoxy was attacked by acetic acid. The result of this attack was that the fiber positions and angles changed, reducing the collection efficiency of the probe. The window provides this protection. The O-rings shown in Figure 6-4 seal the window so that the sample cannot reach the fiber ends.

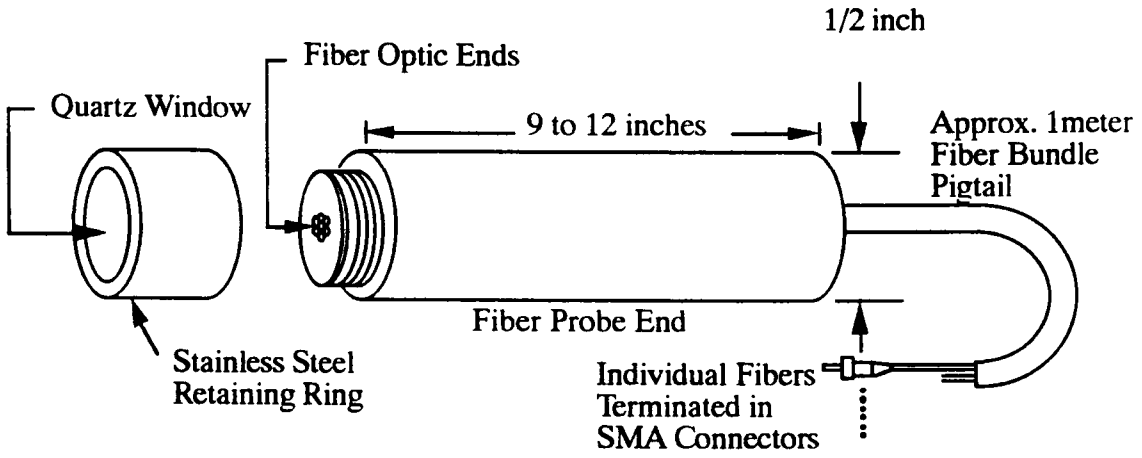


Figure 6-3. Sample Head

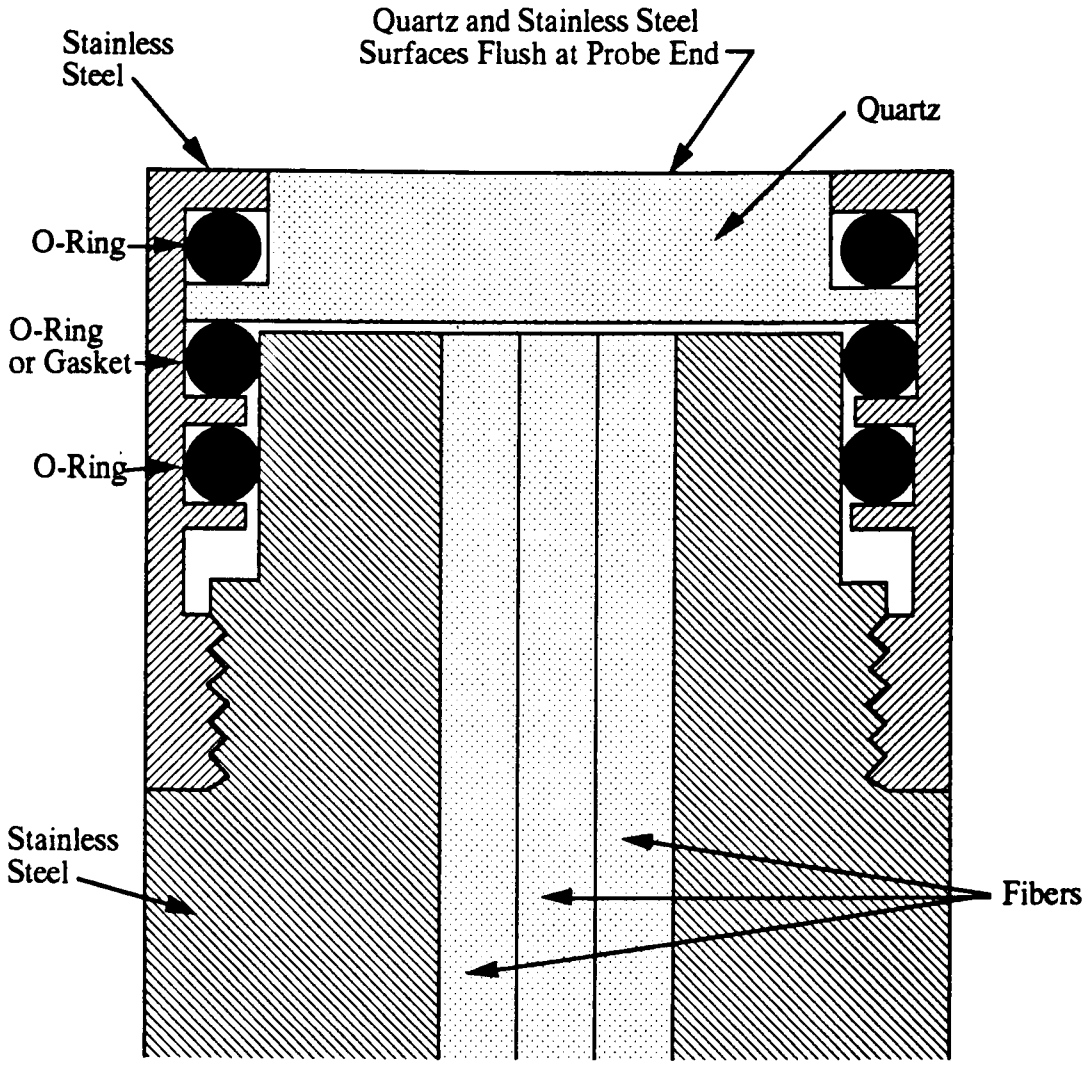


Figure 6-4. Sealing Details

Selection of O-ring material is critical. A probe, of the design shown in Figures 6-3 and 6-4 with Buna-n (rubber) O-rings, was tried with acetic acid. The O-rings were promptly softened, and the window became unseated. Table 6-3 lists some compatibility characteristics of typical chemicals used in this project with two inexpensive O-ring materials. [Parker 77, pp. B10-B27] Neither is satisfactory for the whole range of chemicals. Kalrez, a much more expensive material, resists all the listed chemicals. [DuPont 1980, pp. 18-22] It is the O-ring material of choice for this application.

Table 6-3. O-Ring Compatibility with Sample

	Viton	Buna-n
Acetic Acid (Glacial)	Unsatisfactory	Fair
Acetone	Unsatisfactory	Satisfactory
Benzene	Satisfactory	Unsatisfactory
Cyclohexane	Satisfactory	Unsatisfactory
Eastman Solvent	Unsatisfactory	Fair
Isopropanol	Satisfactory	Satisfactory
Toluene	Satisfactory	Unsatisfactory

The face of the window must be flush with the end face of the stainless steel retaining ring (Figure 6-4). If the window is recessed, bubbles collect in the recess. Bubbles cause reflection of the laser excitation into the receiving fibers, and this adds spurious features to the spectrum.

The Sample Cell

For laboratory testing, the sample cell consists of a 50 ml graduated cylinder with the probe inserted such that the probe tip is at the 25 ml mark. Experimental experience has shown that variations in sampling geometry change the height of the Raman features (Appendix E). Since reproducible chemometrics require consistent line heights, it is necessary to keep the geometry as consistent as possible from one reading to the next. By inserting the probe to the same mark every time, and taking care to keep the probe centered and vertical, highly reproducible spectra are obtained. The 25 ml level represents a compromise. At this level, the probe is far enough from the bottom of the vessel that the vessel artifacts are not significant, but the samples are not so large as to create an inordinate volume of waste.

For application on a distillation column, a sample cell similar to the one shown in Figure 6-5 will be used. [Roberts 90] The sampled chemical stream enters the cell at the bottom through an inlet pipe. The input chemical flow immediately encounters a baffle which diverts the flow toward the field of view of the sample head, and also

On-Line Raman Sample Cell

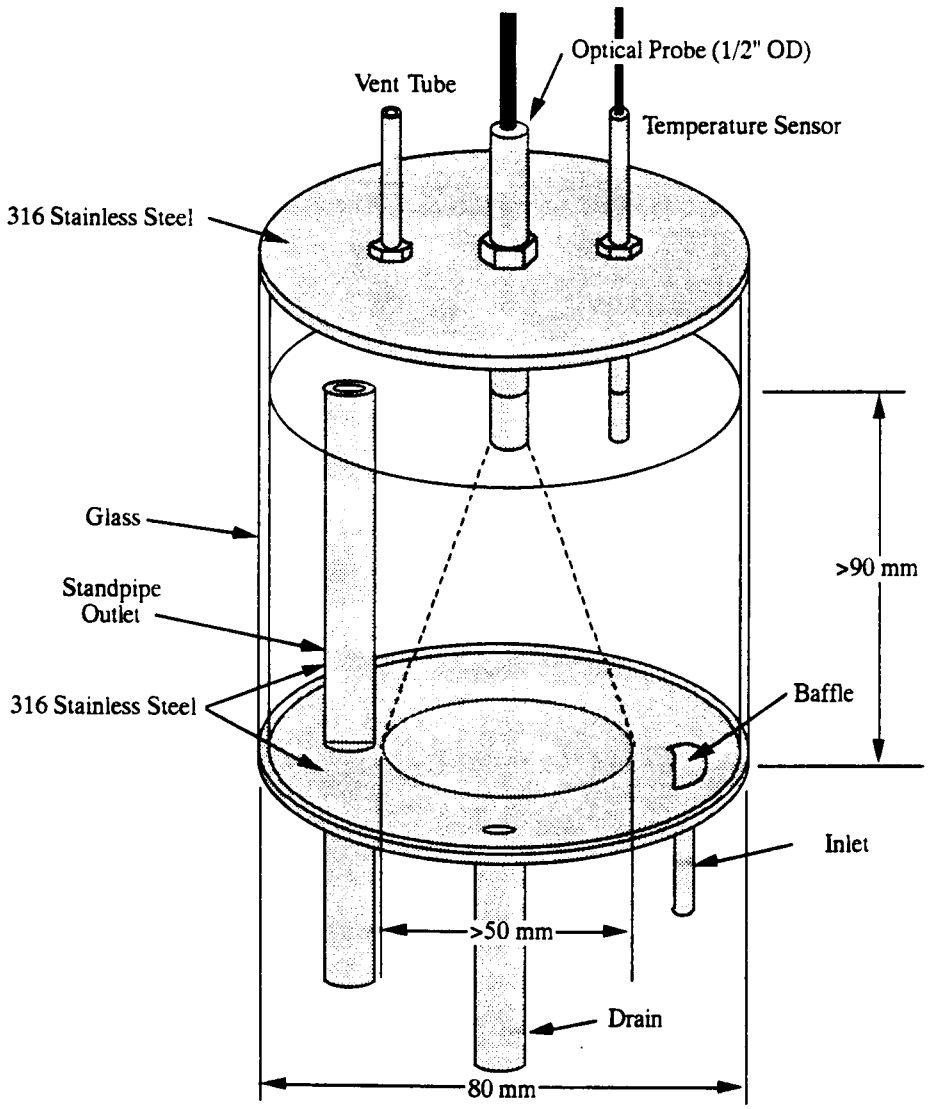


Figure 6-5. Sample Cell

promotes mixing. The level of the sample and, consequently the volume seen by the sample head, is controlled by a standpipe outlet. A vent tube at the top of the cell prevents pressure from building in the cell. A separate drain is included for emptying the cell for cleaning.

The sample head and temperature probe enter the cell through the top. The sample head is pointed vertically downward into the sample volume, and is held in place with a Swagelok fitting. Both the probe model and experimental results show that for the present probe design, the sample head gathers light from a cone about 90 mm in height, and about 50 mm in diameter at the base. A temperature probe is included, although no significant temperature effects on the Raman scattering process are expected.

The cell is shown with a cylindrical glass wall. This is the design that will be used when the system is used with the UT isopropanol column. In order to identify problems during the testing process, a transparent wall will be used. In normal operation, the glass wall will be covered with an opaque jacket to avoid having stray light contaminate the measurement. The rest of the cell is fabricated from 316-series stainless steel.

CHAPTER 7

COUPLING A FIBER BUNDLE TO AN INSTRUMENT

Basic Concepts

Before exploring the technical details involved in coupling the receiving fiber bundle into the interferometer it is desirable to reflect on the major concepts that have bearing on the problem. The instrument has both an area and angle of acceptance. [Webb 89] At a given point along the optical axis of a system, including the input of an instrument, a light ray can be characterized by a vector giving its distance from and its angle relative to the optical axis. [Yariv 85, pp. 17-18]

The most important concept is this. The instrument has both a sensitive area, and an angle of acceptance. This is illustrated in Figure 7-1. It is not difficult to visualize the idea that light rays that hit outside the sensitive area are not detected by the instrument. From this point, it is not a great conceptual leap to consider that if a light ray hits the sensitive area, but is outside the angle of acceptance, it will not propagate all the way through the instrument to the detector. Only the rays that fall within both the area and angle of acceptance are detected.

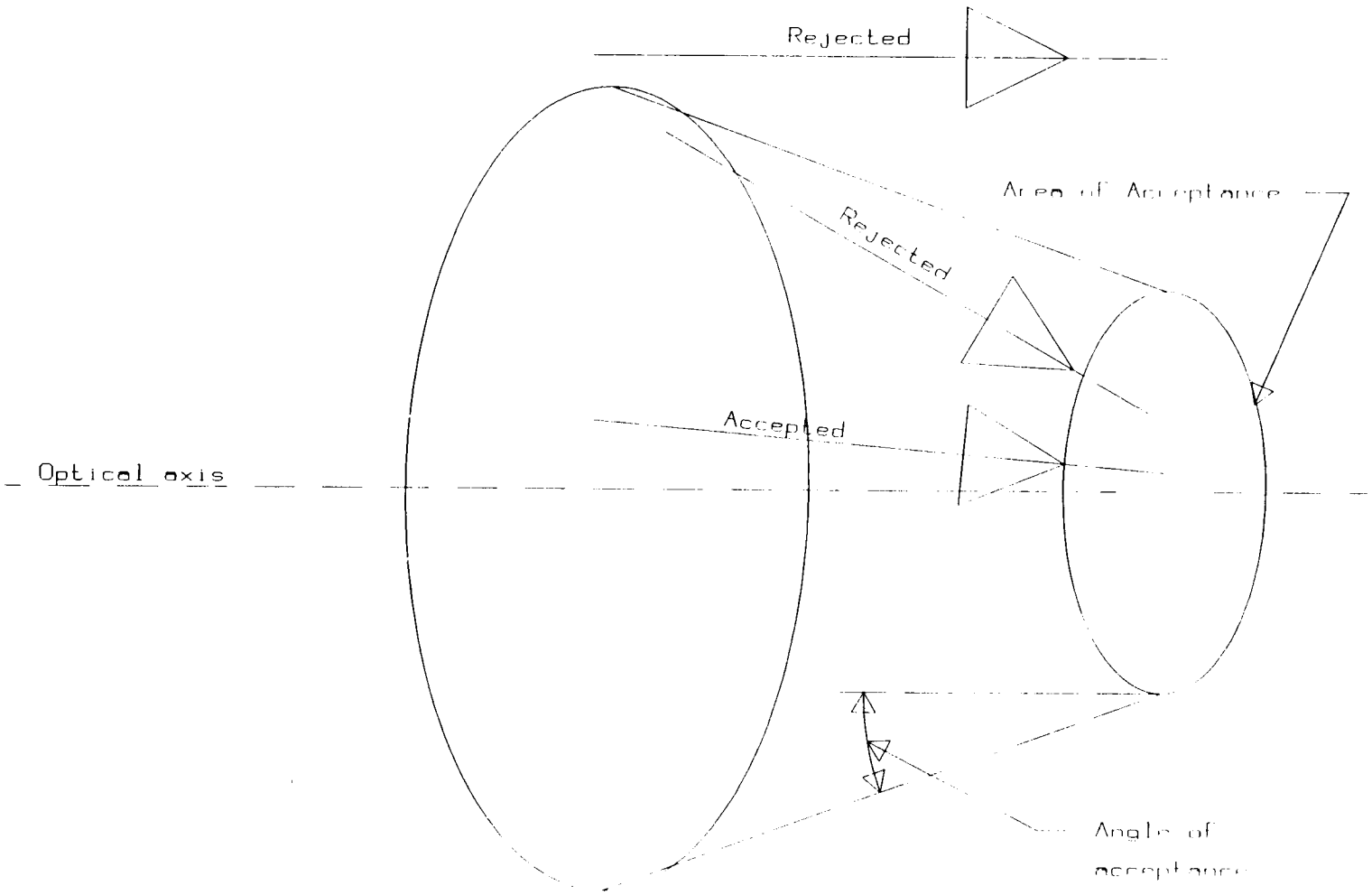
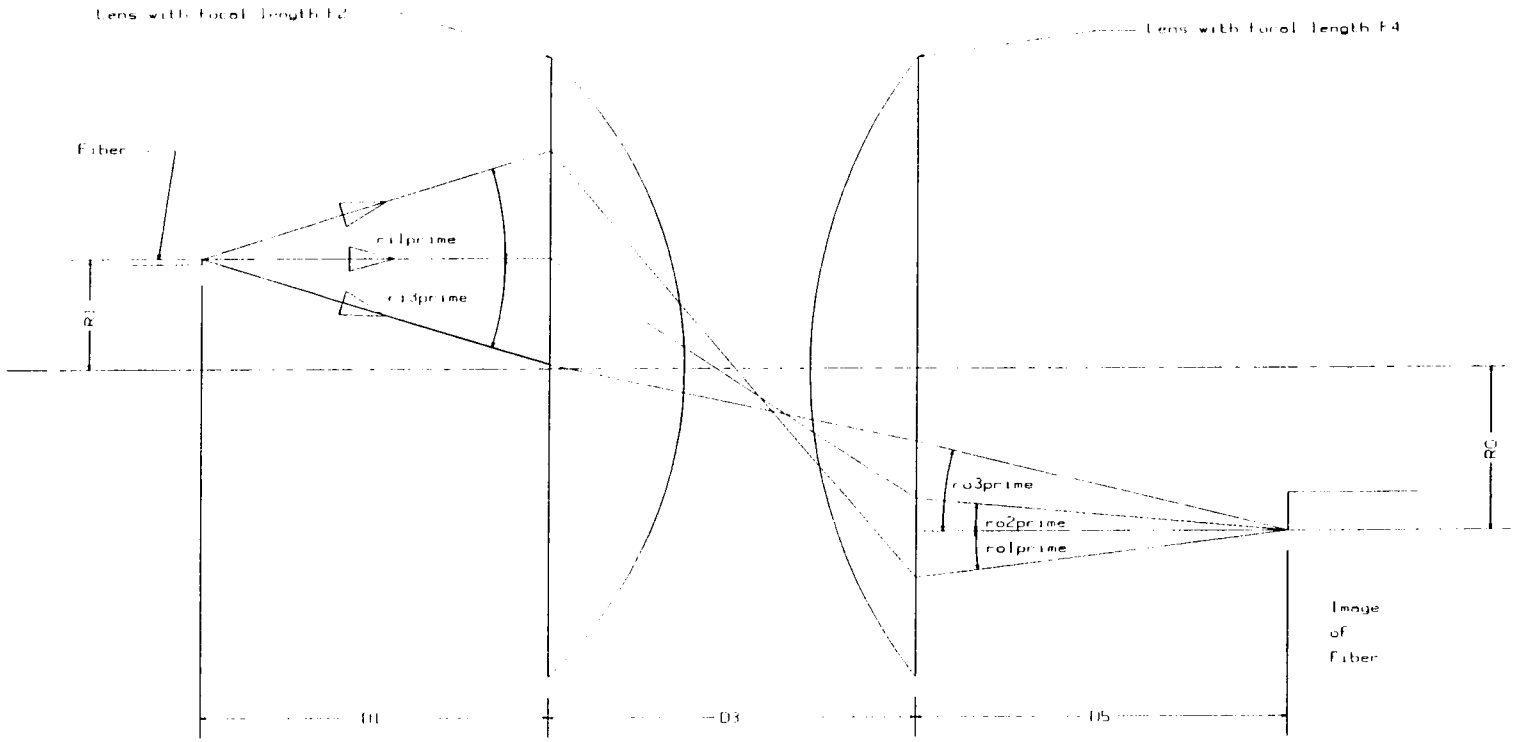


Figure 7-1. Area and Angle of Acceptance

To simplify the consideration, assume that the instrument response has axial symmetry. That is, the sensitive area is within a circle around the optical axis. In addition, the angle of acceptance is the same all the way around. Thus, the only light rays accepted by the instrument are those that fall within the cone.

To simplify the consideration even further, assume that the light source being coupled into the interferometer also has axial symmetry. That is, the light rays emanating from the source form a cone shaped pattern. By assuming axial symmetry of both the source and the target, we are able to visualize a three dimensional problem in two dimensions (Figure 7-2). The rays can be traced on a sectional plane that passes through the system and contains the optical axis.

For the two dimensional problem, at any point along the optical axis, the ray can be completely characterized by a two element vector. The first element gives the distance of the ray from the optical axis; the second element gives the angle of the ray with respect to the optical axis. For an optical system, the input and output rays are each characterized by two dimensional vectors. The optical system is characterized by a matrix that transforms the input vector into the output vector.



Notes:

1. Dimensions must be in consistent units. Usually distances are in millimeters and angles in radians.
2. To couple maximum power from the fiber to its image, R0 must be within the acceptance area and r0primes must be within the angle of acceptance for the interferometer.

Ray Tracing Scheme for Optimizing Receiving Optics

Figure 7-2. Fiber Bundle to Instrument Coupler

The essential problem in designing optical couplers is to devise a system that captures the rays emanating from the source, and transforms all of them into rays acceptable to the instrument. Mathematically, this is accomplished by selecting an optical system that is characterized by a matrix that will transform the worst case source ray into a ray acceptable to the target.

Imaging

A straightforward way to design a coupler is by imaging. That is, the optics are designed in such a way that all the rays that emanate from a single point on the source converge to a single point on the target. The assumption of imaging imposes a constraint. Five parameters, three spacings and two focal lengths, completely describe the coupler shown in Figure 7-2. Any four may be arbitrarily selected, but if imaging is assumed, the fifth parameter will be a function of the other four.

The image analysis as it is presented in textbooks has two steps. First, given four of the five parameters, find the fifth one. [Hecht 87, pp. 145-149] Next, given the radius and angle of a source ray, and the five parameters, calculate the radius and angle of the image ray at the

target. [Yariv 85, p. 19] The problem is a tedious but straightforward application of ray matrices.

Ray Matrices

An object may be visualized as a source of infinitely many rays, but the rays are restricted to having certain properties. A meridional ray at a given position along the optical axis is characterized by the vector $[r,rprime]^T$, where r is the radius, or distance of the ray from the optical axis and rprime is the slope of the ray with respect to the optical axis. The ray traces are illustrated in Figure 7-2. For this analysis it is assumed that an object is characterized as having an continuum of rays whose properties are bounded by:

$|r_o| <= r_{omax}$ and

$|rprime_o| <= rprime_{omax}$

An image is characterized as accepting all rays whose properties are bounded by

$|r_i| <= r_{imax}$ and

$|rprime_i| <= rprime_{imax}$

The most difficult ray to image is the outermost one. If the object ray with the properties $[r_{omax}, rprime_{omax}]^T$ can be transformed into a ray with the properties $[r_i, rprime_i]^T$ and $|r_i| <= r_{imax}$ and $|rprime_i| <= rprime_{imax}$ then the outermost object ray will be included in the image. If this condition can be met, then all the power emanating from the object will be coupled into the image.

In order to simplify the analysis, three assumptions are made. The system is assumed to be symmetrical about the optical axis. Skew rays are ignored. The paraxial approximation applies.

An optical element is viewed as a transformer that transforms the object ray into an image ray. Since both the object and image rays are characterized by two dimensional vectors, it is not surprising that the optical element is characterized by a 2X2 matrix. In its most general form the transformation can be described as follows.

$$\begin{bmatrix} r_i \\ rprime_i \end{bmatrix} = \begin{bmatrix} e_1 & e_2 \\ e_3 & e_4 \end{bmatrix} \begin{bmatrix} r_o \\ rprime_o \end{bmatrix}$$

The two optical elements that are of interest in coupling a fiber to an interferometer are the air gap and

the thin converging lens. For an air gap of axial length d, the ray matrix is M_a. For a thin converging lens of focal length f, the ray matrix is M_l [Yariv 85, p. 19].

$$M_a = \begin{bmatrix} 1 & d \\ 0 & 1 \end{bmatrix}$$

$$M_l = \begin{bmatrix} 1 & 0 \\ \frac{-1}{f} & 1 \end{bmatrix}$$

If an optical system is made up of several optical elements, then its ray matrix is the product of the matrices of the individual elements. For the system shown in Figure 7-2:

$$M = \begin{bmatrix} 1 & d_5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{-1}{f_4} & 1 \end{bmatrix} \begin{bmatrix} 1 & d_3 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{-1}{f_2} & 1 \end{bmatrix} \begin{bmatrix} 1 & d_1 \\ 0 & 1 \end{bmatrix}$$

Synthesis of an Optical Coupler

The synthesis of an optical coupler is the inverse of the ray matrix analysis problem. Given the worst case source ray, and the range of acceptance of the target, what set of the five optical parameters will transform the source ray

into a ray whose radius and angle are acceptable to the target? A major difficulty with this problem is that it may be impossible to solve. [Webb 89]

The difficulty is caused by the principle of optical invariance. A change in the imaging optics will always cause size of the image to change in one direction while the ray angle changes in the other. For example, suppose the source is imaged entirely within the sensitive area of the target, but the angles of some of the image rays are larger than the angle of acceptance. Suppose the optics are then changed to assure that all the image rays fall within the angle of acceptance. This incidentally magnifies the image. Unfortunately, there is a good possibility that the change will magnify the image so much that it becomes larger than the area of acceptance.

It may be that any set of optics that makes the image small enough to stay within the area of acceptance, causes some of the rays to fall outside the angle of acceptance. It may also be that any set of optics that keeps all the image rays within the angle of acceptance always produces an image larger than the area of acceptance. In technical terms, there may be an etendue mismatch.

Etendue

What happens if the simplifying assumption of axial symmetry is not made? In general the source or the target consists of a region of points, each of which can emit light at or be responsive to light over a variable range of angles. If the points in some area, S, emit light into a solid angle, Q, then the increment of light emitted from a differential area, dS, into a differential solid angle, dQ, is $d^2G = dSdQ$. All the light emitted by the source is the sum of the incremental parts. Thus, $G = \int dSdQ$ is the total light emitted by the source, or the geometric extent, or the etendue. [Lerner 88, p. 29] A fiber whose critical angle is Ω emits light into a cone with a half angle of Ω. Integrating over the cone, $G = \pi * S * \sin^2\Omega$, is the etendue for a single optical fiber. The light emitting ability and the light gathering ability of an optical element are the same. Either can be viewed as the etendue.

For a system of multiple elements, the etendue of the system is the smallest etendue of any of the elements. In a sense, the etendue is the maximum beam size that will be accepted by the system. An optical transformer, such as a lens, will change the solid angle of a beam in one sense, and the area in the opposite sense, but it will not change the etendue. The practical implication is that if a source

has a larger etendue than the target, it is not possible to couple all the optical power from the source to the target.

The Invariant

Since the calculation of etendue for all but the simplest of shapes is rather involved, a simpler (but more conservative) way to test for the efficiency of coupling is used in this application. The invariant is the product the radius and angle of the outermost ray of an optical element [Webb 89]. If the invariant of the source is less than or equal to the invariant of the target, then the principal of optical invariance does not preclude the possibility of coupling all the power from the source to the target.

The specific problem in this application was to couple a bundle of six fibers to a Perkin-Elmer 1700 interferometer. The fibers have 400 micron core diameters and a numerical aperture of 0.225. They are arranged in a regular hexagon about the optical axis with the outside edges of the cores being 0.8 mm from the optical axis. The response of the interferometer is not symmetrical about the optical axis. In its most restrictive direction, the angle of acceptance is 0.0754 radians and the radius of acceptance is 2.8 mm. [Bennett 89] Thus, a conservatively designed coupling system would be one which transforms a ray with a radius of

0.8 mm and an angle of 0.225 radians into a ray with a radius of less than 2.8 mm and an angle of less than 0.0754 radians.

Is such an arrangement possible? The invariant of the fiber bundle is 0.8*0.225=0.18. The invariant of the interferometer is no less than 2.8*0.0754=0.21. Therefore, the principle of optical invariance does not preclude the coupling of all the energy from the fiber bundle to the interferometer. One arrangement that would couple all the power from the fiber bundle to the interferometer is d_1=46.08, f_2=100, d_3=68, f_4=100, and d_5=287 mm. This transforms a ray with a radius of 0.8 mm and an angle of 0.225 radians into a ray with a radius of 2.77 mm and an angle of 0.0754 radians.

This arrangement has not been implemented, as it would require significant modification of the existing equipment. The arrangement actually used to produce the spectrum shown in Figure 7-3 is d_1=63.63, f_2=100, d_3=25, f_4=100, and d_5=200 mm. This transforms a ray with a radius of 0.8 mm and an angle of 0.225 radians into a ray with a radius of 2.2 mm and an angle of 0.0958 radians. It is estimated that this arrangement has a power loss of no greater than 5% due to angular overfill.

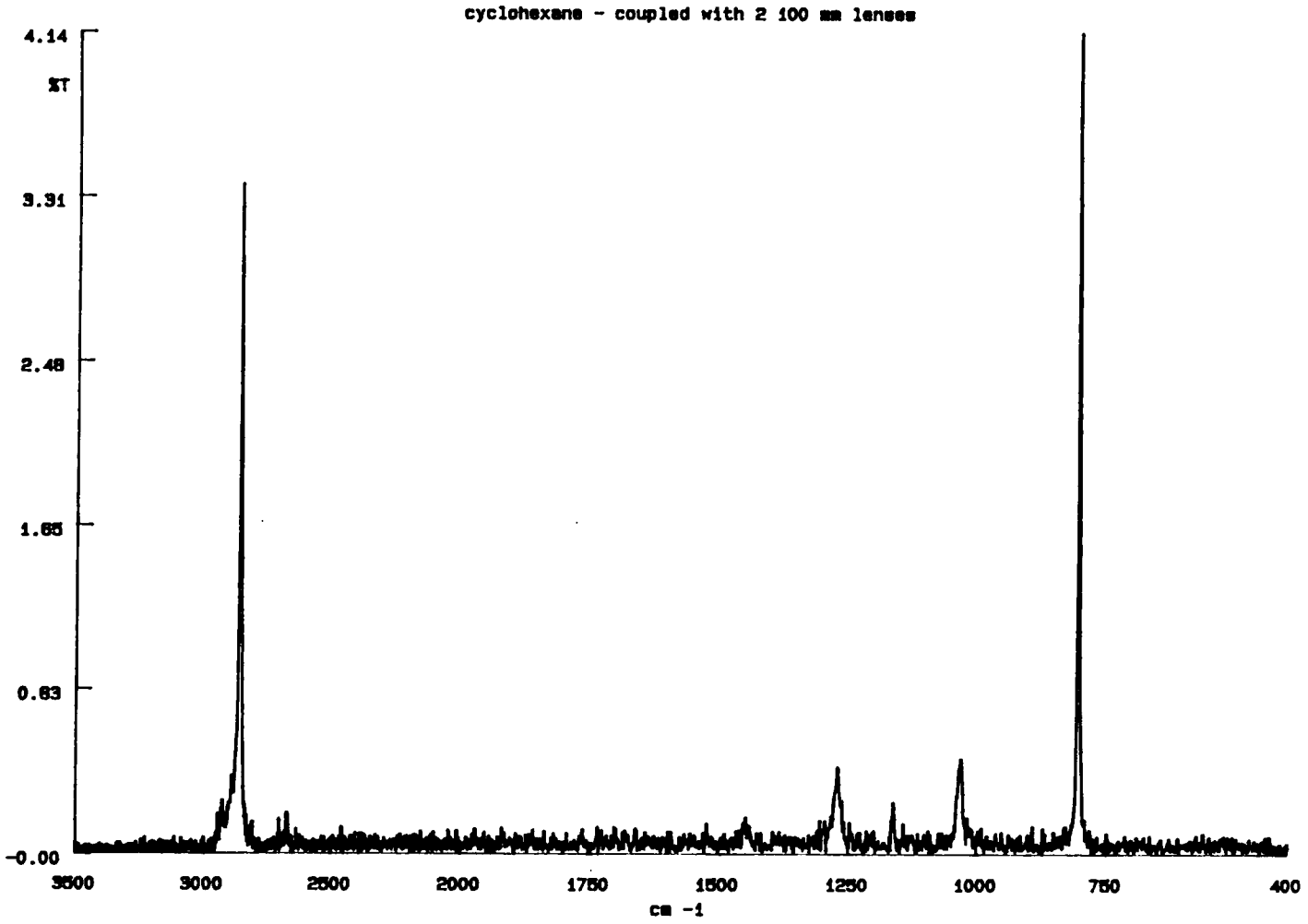


Figure 7-3. Cyclohexane Through Optimal Coupler

Design Software

Several computer applications applying these principles have been developed. One is a Lotus Symphony spreadsheet on the IBM-PC for ray matrix analysis. In this application the user specifies the focal lengths of the lenses, the distance between them, and the distance from the second lens to the Jacquinot stop of the interferometer. The spreadsheet calculates the distance from the end of the fiber bundle to the first lens to assure that an image forms. The user also specifies three rays emanating from the object. The spreadsheet calculates the properties of the corresponding three image rays. This spreadsheet is useful for determining whether the image rays resulting from a specified object and a specified coupling system fall within the limits of acceptance of the receiving element.

The design of a coupling system is somewhat more involved. Given the characteristics of the outermost ray emitted by the object, and the outermost ray accepted by the receiving element, the problem is to find a combination of d_1, f_2, d_3, f_4, and d_5 that will cause all the rays emitted by the object to be accepted by the receiving instrument. To address the design problem, a VAX Fortran application has been developed. The source code is listed in Appendix I. In this application, the user specifies properties of the

outermost object and image rays, and ranges for f_2, d_3, f_4, and d_5. The program computes the image ray for each combination of parameters. If the image ray falls within the bounds of the outermost acceptable image ray, its properties and parameters are stored to an output file. If an image ray is outside the bounds, then the sum of its squared differences with the acceptable image ray is computed. If no set of parameters results in an acceptable design, then the design with the smallest sum of squared differences is stored in the output file. This gives the user an indication of how to specify ranges on a successive run.

Observed Performance

The coupling efficiencies of several different schemes have been measured. Several parameters were kept fixed, specifically: d_3=25, f_4=100, and d_5=200. Lenses of different focal lengths were used at f_2. The distance, d_1, was adjusted for maximum coupling. For all the different lenses, a spectrum of cyclohexane was taken. All parameters (such as excitation power to the sample) were kept constant, except for the optical coupling. The height of the cyclohexane features is proportional to the power being coupled into the interferometer. The results are summarized in Table 7-1. Cyclohexane intensity is the height of the

feature at 805 wave numbers. Noise is the maximum intensity in the 2600 to 1500 wave number band. Since the 100 mm lens provided the best coupling, it was used as a standard of comparison in the table.

Table 7-1. System Performance for Several Couplers

F2 Focal Length	Cyclo-hexane Intensity	Noise Level	Comparative Throughput	Signal to Noise Ratio
125 mm	2.77	0.11	-1.74 dB	14.01 dB
100 mm	4.14	0.15	0.00 dB	14.41 dB
75 mm	3.30	0.14	-0.98 dB	13.72 dB
50 mm	2.08	0.15	-2.99 dB	11.42 dB
50 mm	0.86	0.19	-6.83 dB	7.58 dB

(Note: The second 50 mm lens is the one furnished with the interferometer. It has a prism cemented to its center)

CHAPTER 8

CONTROL SOFTWARE

The Programming Environment

The software to control the on-line analyzer is non-portable and hardware dependent. Some of the interface functions between the computer and the scientific apparatus are demonstrated in prototype programs. Source codes for prototype programs are listed in Appendix J. Source codes for the control programs are listed in Appendix K. Both the control and the prototype software will only work on an IBM-AT compatible whose expansion board switches have been set to conform to the I/O port addresses shown in Table 8-1. The parallel printer port must be used to control the beam dump solenoid. There must be two Daedal PC-23 motor controller boards installed. Each of these must be connected to a three axis positioner with encoders and limit switches. There must be a National Instruments PCII IEEE-488 interface board connected to a Perkin-Elmer 1700 interferometer. In addition, to simplify development, the C language programs make extensive use of Microsoft and National Instrument C library functions. The chemometrics program must be compiled with the Microsoft Fortran Optimizing Compiler.

I/O Map

Computer 0 (UT Tag Number 399509) has had all the hardware installed to act as the composition analyzer control computer. I/O addresses have been set per Table 8-1.

The printer port, I/O address 378-37F, is used to control the beam dump solenoid. Messages to the process control computer will be sent through communications port 3, 3E8-3EF. If a printer is needed it will be connected to communications port 1, 3F8-3FF.

Positioner Programs

Each three axis positioner has its own control system. Each axis has a motor under user control. Each axis has a position encoder that can be queried by the user. Each axis has home and limit switches whose function is controlled by the user. Each positioner has its own microprocessor based controller. Commands are sent to the positioner in the form of ASCII strings sent through an I/O port of the control computer. [Daedal 88]

The prototype program DRIVER.C, listed in Appendix J, demonstrates all the basic functions needed to control the positioner. First, it resets the positioner. Then, it sets

Table 8-1. Control Computer I/O Map

I/O Hex Address	Used by	Source
000-0FF	System Board	[Greenleaf 89b]
1F0-1F8	Fixed Disk	[Greenleaf 89b]
200-207	Game Port	[Greenleaf 89a]
278-27F	Parallel Printer Port 3 (IRQ 5)	[Greenleaf 89a]
2B8-2BF	IEEE-488 Bus (DMA 1 IRQ 7)	[National 89a]
2E8-2EF	Serial Comm Port 4	[Greenleaf 89a]
2F8-2FF	Serial Communications Port 2	[Greenleaf 89a]
300-303	Motor Controller 0 (IRQ 3 IRQ 4)	[Daedal 88]
310-313	Motor Controller 1 (IRQ 3 IRQ 4)	[Daedal 88]
360-36F	Reserved	[Greenleaf 89b]
378-37F	Parallel Printer Port 2	[Greenleaf 89a]
380-38F	SDLC Port 2	[Greenleaf 89b]
3A0-3AF	SDLC Port 1	[Greenleaf 89b]
3B0-3BF	Monochrome/Printer Adapter (IRQ 7)	[IBM 84b]
3C0-3CF	Enhanced Graphics Adapter (IRQ 2)	[IBM 84a]
3D0-3DF	Color Graphics Adapter	[Greenleaf 89b]
3E0-3EE	A/D Convertor (Interrupts Off)	[Singular 89]
3E8-3EF	Serial Comm Port 3	[Greenleaf 89a]
3F0-3F7	Floppy Disk Controller	[Greenleaf 89b]
3F8-3FF	Serial Comm Port 1 (IRQ 4)	[Greenleaf 89a]

up and executes a move, queries the positioner for the new coordinates, reverses the move, and queries the positioner for final coordinates.

The prototype program RUNXKEYA.C, listed in Appendix J, takes strings interactively from the keyboard and sends them to the positioner. With this program, the user can develop a sequence of commands, and verify that it produces the desired responses from the positioner. Once a sequence of commands is developed and verified, it can then be included other programs.

There several pitfalls to avoid when using RUNXKEYA.C. These arise out of the fact that some commands cause the positioner to issue response data, and some do not. For example, the "A" command sets acceleration for an axis, but does not produce a response. If a query for a response is issued right after a non-responding command, the program will lock up, waiting for the response to a command that cannot be issued. On the other hand, if a command, such as PR, does issue a response, it should be queried immediately. The positioner will not process the next command until the current response is cleared from the buffer. It is the responsibility of the user to know which commands issue a response and which do not. Commands that issue a response

are listed as "Status" commands in the positioner documentation. [Daedal 88]

Frequently Used Positioner Commands

Each axis of the positioner has a home switch and limit switches. The home switch is at the middle of travel marks the zero coordinate position. The limit switches mark the ends of travel. The positioner controller ignores the home and limit switches unless the user specifically activates them. The command LD activates the switches. For example, 1LD0 activates both limit switches for axis 1.

It is often desirable to start a sequence of positioner operations by seeking the "home" or zero coordinate position. GH is the "go home" command. For example, 3GH+5 sends axis 3 home, at 5 revolutions per second in the clockwise direction. Note that it is unwise to send the positioner home too fast. A velocity of 5 revolutions per second is a good value. [Newell 90] If the positioner starts to seek home in the wrong direction, it will bounce off the end limit, reverse direction, and continue seeking. To find out if the homing operation was successful, the RC command is used. For example, 1RC checks whether or not the last "go home" attempt on axis 1 was successful. A response message of "*@" means that the attempt was successful.

The PR command reports the absolute position with respect to home. For example, in response to the command 1PR, the positioner might reply 1:100000. This means that axis 1 is 100000 steps clockwise from home.

The travel parameters are written to a buffer and remain in force until changed. For example, 1A10 orders axis 1 to use an acceleration of 10 revolutions per second per second in its movements, 2V10 orders axis 2 to use a maximum velocity of 10 revolutions per second in its movements, and 3D10000 orders axis 3 to use a displacement of 10000 steps on its next move. The positioner will not execute a move along a given axis until the buffer holds non-zero values for the acceleration, velocity, and displacement along the axis.

The velocity profile for a given movement is trapezoidal. The velocity will increase linearly at the prescribed rate of acceleration until the prescribed velocity is reached. Movement will continue constant at the prescribed velocity until it is time to decelerate. Velocity decrease is linear at the prescribed acceleration rate. The profile is computed such that accomplishes the prescribed amount of displacement. If the displacement is so short that the prescribed velocity is never reached, then the velocity

profile is triangular. The velocity profile is handled internally by the microprocessor without intervention by the user.

Displacement of the next move can be absolute or incremental. Absolute displacement is the displacement with respect to home. Incremental displacement is the displacement with respect to the current position. For example, suppose the buffer contains 1D25600. If the axis is placed in the absolute mode with the command 1MPA, then the next move will move axis 1 to a position 25600 steps from the home position. If the axis is placed in the incremental mode with the 1MPI command, the next move will move axis 1 to a position 25600 steps from the current position.

To cause the positioner to actually move, it is necessary to issue a "go" command. As already noted, non-zero acceleration, velocity, and displacement parameters must be residing in the buffer before the positioner will move in response to a "go" command. For single axis movements, the axis number precedes the G. Thus, 1G means to go along axis 1. For multiple axis movements, the axis numbers follow the G. Thus, G12 means to trace a diagonal line in the 1-2 plane.

Positioner Commands in Control Software

In the on-line analyzer, two positioners are used. One moves the end of the selected excitation fiber into the focus of the input coupling device. The other moves the selected receiving fiber bundle into the object position of the output coupler. The operation of a given positioner can be viewed as initially setting up the operating parameters and mode, then repeatedly cycling through a sequence of several absolute displacements until interrupted by the user. The following operations are included in the CYCLER4.C program listed in Appendix K.

There is a setup portion of the control program. First, the initialize function is used to assure that the positioner is ready to receive commands. Next, the limit switches are activated, and all axes are sent home. Then, the homing operation is checked for success. If the positioner fails to initialize, or fails to go home, then the program issues warning messages and returns to the operating system command level. If the positioner initializes and goes home, then acceleration and velocity parameters are written to the buffer, and the axes are set in absolute displacement mode.

Once the positioners are set up, there is cycling portion of the control program. For the selected cell number, the three absolute coordinates are read from an array, concatenated into command strings, and written to the positioner. The "go" command then moves the positioner to the absolute displacement.

Interface with Beam Dump Solenoid

The beam dump is a block of metal that is used to block and absorb the full power of the laser beam while the excitation fiber ends are being moved by the positioner. The purpose of the beam dump is to prevent random reflections of the laser beam, and to prevent damage to the excitation fiber end connectors, during positioning operations.

The beam dump is mounted on a solenoid. When the solenoid is de-energized, the beam dump falls by gravity to the blocking position. When the solenoid is energized, it pulls the beam dump out of the blocking position and allows the laser beam to hit the microscope objective. The solenoid is energized by 24 volt DC power that is controlled by a solid state switch that is controlled by TTL logic. The input of the TTL logic of the solenoid switch is connected to pin 2 of parallel printer port 2.

The printer port is at I/O address 378 hex. [Greenleaf 89a] The data pins of the adapter are pins 2 through 9. The logic states of the pins 2 through 9 correspond to the logic states of bits 0 through 7 respectively of the last byte written to address 378 hex. The states of the pins remain latched until the port is written by another command. [IBM 84b, pp. 1-7]

The state of pin 2 is controlled by the state of the least significant bit of the byte written to address 378 hex. Thus, if a 1 is written to the port, pin 2 is latched high. If a 0 is written to the port, pin 2 is latched low. This was tested by measuring the voltage on pin 2 of the parallel printer port with a volt meter. After a 1 is written to address 378 hex, the voltage latches to 5.11 volts. After a 0 is written to address 378 hex, the voltage latches to 0.0 volts.

Interface with Interferometer

The interface between the control computer and the interferometer is handled by an IEEE-488 bus. The details of the interface are not included in this thesis due to a lack of technical data from the vendor.

Control Software

The control software for the on-line analyzer was developed in Microsoft Quick C and Microsoft Fortran. It is designed to run on an IBM-AT compatible under MS-DOS. The software conforms to the specification in Appendix B. Source codes for the laboratory test version of the control program, CYCLER4.C, the spectrum collection program, SPECR0.C, and the chemometrics training program, CHEMO.FOR, are listed in Appendix K.

The data flow diagram for the control software is shown in Figure 8-1. The spectrum program is used to obtain calibration spectra for mixtures of known composition in a specified sample cell. The sample cells are numbered 0 through 4. The program reads the description of the cell from the appropriate cell description file, cella0.dat through cella4.dat. The program also reads the description of the spectra to be collected from the spectrum description file, spec0.dat. The program collects spectra from the interferometer via the IEEE-488 bus. It saves each spectrum in a file whose name is listed in spec0.dat.

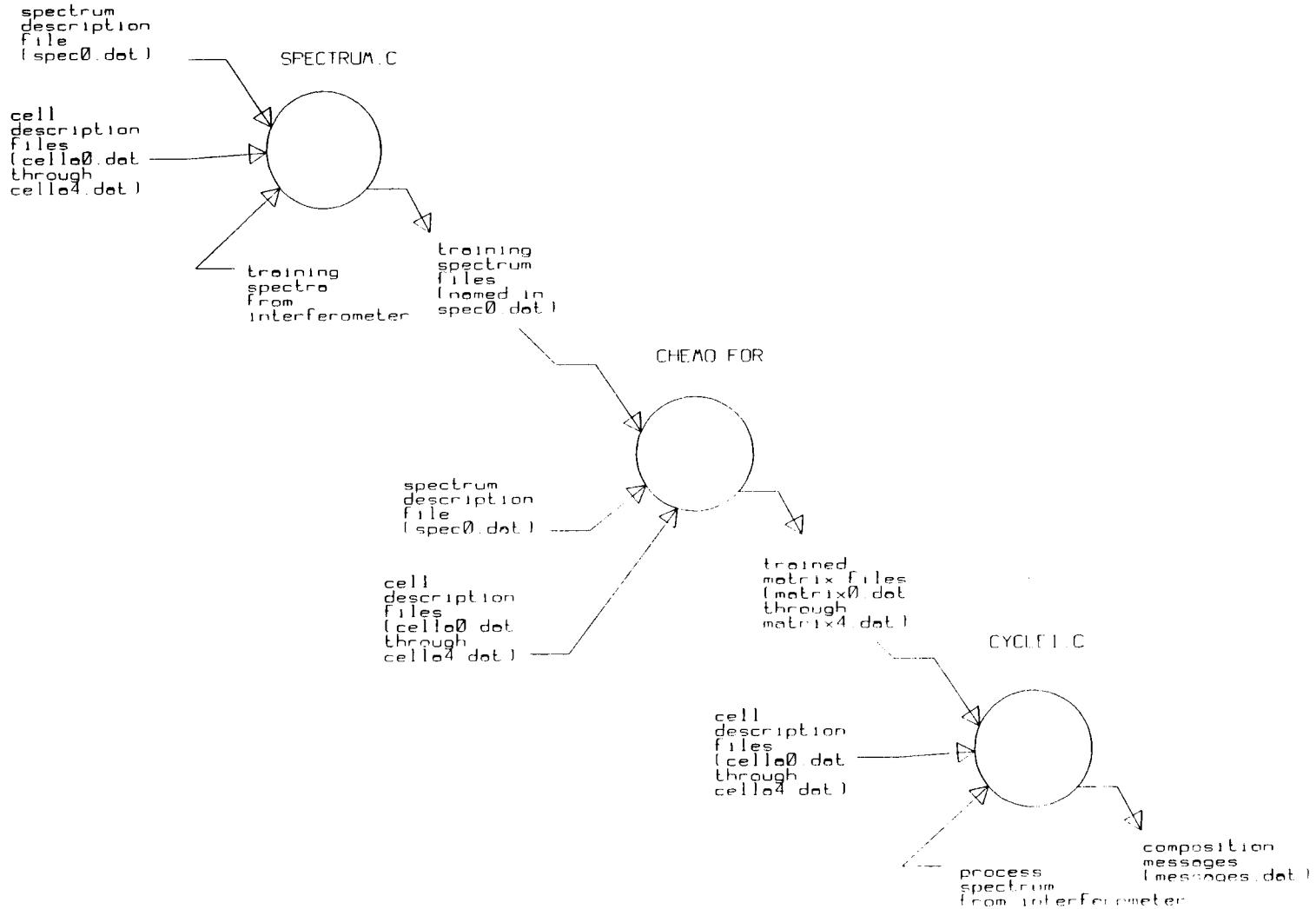


Figure 8-1. Data Flow Diagram of Control Software

The chemometrics program reads the spectrum description and composition vectors from the spectrum description file, the cell description from the cell description file, and the spectrum vectors from the spectrum files. From these data the program calculates the data needed for calibration matrix file. The calibration matrix file contains the number of training samples, the eigenvectors of the product moment matrix, the eigenvalues of the product moment matrix, the column retention vector, the load selection matrix, and the regression matrix. The data are saved to one of the matrix files matrix0.dat through matrix4.dat. The transformation of the set of spectra in the spectrum files into the calibration matrix data is accomplished by the techniques described by Fredericks. [Fredericks 85b]

The cycle program determines the composition of an unknown mixture in the sample cells. It reads cell descriptions from the cell description files and the analysis matrices from the matrix files. It then cycles through each of the active sample cells. When the initial implementation is complete, it will collect a spectrum vector, use the spectrum vector and the data from the analysis matrix file to obtain the composition vector, and then write the composition message to the messages.dat file. In a future implementation, the program will send the

composition message to the process control computer via an RS-232 link.

There are two modes of operation for this system, a calibration mode, and a monitor mode. The calibration mode occurs only at initial setup, or when some change to the instrument requires that a sample cell be recalibrated. In calibration mode, a set of calibration mixtures of known composition is prepared in advance. The spectrum program is used to collect a spectrum of each mixture in the sample cell being calibrated. After all the spectra are collected, the chemometrics program transforms those spectra flagged as training spectra into calibration data for the sample cell. The calibration data are stored in a file, and are used as the basis for analysis in the monitor mode.

The chemometrics program also provides verification services. That is, the performance of the system is verified before it is used to analyze unknown mixtures. Several of the spectra collected by the spectrum program are flagged as verification samples. That is, they are not used in the computation of the calibration data, but rather, are used to verify its predictions. The chemometrics program reads the cell descriptions and spectra, and computes and reports the compositions and errors for each spectrum flagged as a verification spectrum.

Routine operation of the instrument is done in the monitor mode. In this mode the cycle program repeatedly cycles through all the active sample cells, determining the composition of the unknown mixture in each. Since the mixture can change over time, it monitors continuously until halted by the user.

User Interface

The analyzer control software is designed for minimal user interaction at runtime. The spectrum program queries the user, after it collects each spectrum, whether to save this version of the spectrum, or to redo the spectrum for this sample. At the end of each operating cycle, the cycle program polls the key buffer to determine if the user has typed Ctrl-X, the quit signal. The chemometrics program querys the user whether or not to report full diagnostics. All other input data to the system are communicated via the data files.

The cell description files, cella0.dat through cella4.dat, contain data required by the spectrum and cycle programs. This is an ASCII file prepared in advance of the run by the user with a text editor or a word processor set for ASCII output. A typical listing for cella0.dat is shown

in Figure 8-2. The first line is the cell flag, an integer either 0 or 1. If the cell is inactive, the flag is 0. In this case, no further entries are required.

```
1
12345 23456 34567
54321 43210 32100
16
0.0
5
8600
8599
8598
8597
8596
3
acetic acid
solvent
water
```

Figure 8-2. Typical Cell Description File

For active cells, the next lines contain the positioner coordinates. First are the coordinates for positioner 0, the excitation positioner. These are a set of three long integers, delimited by spaces, that represent the coordinates in motor steps with respect to the home position for each of the three axes. The first entry is the axis 1 coordinate. The second entry is the axis 2 coordinate. The third entry is the axis 3 coordinate. The next line consists of a similar set of coordinates for positioner 1, the receiving end positioner. The next entry is the number of scans used by the spectrometer when it collects the spectrum. It is an integer. Note that this is the only

spectrometer parameter set by the user from the PC based control software. All other parameters, such as resolution, are set in advance of the run directly with the spectrometer controller keyboard.

The next entry is the normalization number. It is a floating point number, and is used only by the cycle program. If the sample cell mixture has only one component, then this entry is the expected peak height of the calibration standard. During program execution, the expected peak height is compared with the observed peak height to determine the normalization factor. If the sample cell mixture has more than one component, and this entry is less than 0.1, then no normalization is used. If the sample cell mixture has more than one component, and this entry is greater than or equal to 0.1, then the spectrum is normalized by multiplying each component in the spectrum vector by the normalizing factor. This feature is provided to enable the user to compensate for line height drift, if desired.

The next entries are integers that describe the spectral features of interest. The first number is the count of the number of elements in the spectrum vector. The next set of integers are the spectrum vector components. Each entry is a spectrum wavelength in absolute infrared wave numbers (not

the Raman offset from the excitation wavelength). All spectrum wavelengths that are to be used in the analysis are included in this list. Each must appear on a different line. The number of spectrum vector components must agree with the count.

The next integer is the count of the number of components that make up the mixture in the sample cell. A count of 1 is permitted. A single component mixture always has an analysis of 100% of the single component. It is used as a calibration sample. More than one calibration sample is permitted. The normalizing factor is based on the most recent calibration sample encountered. Calibration samples are permitted, but are not required. A count of more than 1 is permitted, but the total count must not exceed the value declared for the constant MAXCOMPS in the declarations of the source codes.

The final entries the cell description file are the component strings. Each string must be entered on its own line, without quote marks. A string may be made up of multiple words. It is the user's responsibility to assure that the sequence of the composition strings agree with the sequence of the components of the composition vectors in the spec0.dat file.

The spectrum description file, spec0.dat, contains the data required by the spectrum and chemometrics programs. This is an ASCII file prepared in advance of the run by the user with a text editor or a word processor set for ASCII output. A typical listing for spec0.dat is shown in Figure 8-3. The first line is the cell number, an integer between 0 and 4. The next line is the number of spectra to be used in the calibration. It is an integer. The next line is a string, without quotes, indicating the filename in which the spectrum is to be stored. The name must conform to the normal MS-DOS naming rules. The next line is the training verification flag. Each sample is either a training sample, used to compute the analysis data, or a verification sample, used to check the predictive accuracy of the calibration data. The flag is an integer. If it is 0, the sample is a verification sample. If the flag is any non-zero integer, the sample is a training sample. The next lines contain the components of the composition vector for the sample. Each component is a floating point value on its own line. The number of elements in the vector must agree with the number of components given in the cell description file. The numbers in the composition vector represent the percentages of each component. The components in a composition vector should total to 100. The sequence of the composition elements must be the same as the sequence of the composition strings in the cell description file, and must be the same

```
0
10
saw01.dat
1
48.838
50.073
1.089
saw02.dat
0
56.898
41.684
1.418
saw04.dat
1
40.584
51.172
8.244
saw05.dat
1
37.476
58.587
3.937
saw06.dat
0
62.506
32.582
4.912
saw08.dat
1
32.254
60.119
7.627
saw09.dat
1
46.309
50.038
3.653
saw10.dat
1
26.922
70.149
2.929
```

Figure 8-3. Typical Spectrum Description File

for each composition vector. The number of spectrum files and composition vectors must agree with the count on the second line of the spectrum description file.

The regression data files, matrix0.dat through matrix4.dat, are lists of numbers that represent the regression data for each cell. They are generated by CHEMO.FOR in the calibration mode, and used as input by CYCLE1.C in the monitor mode. Note that this is handled automatically by the software, and the matrix files must not be edited by the user.

The programs are run by invoking the executable filenames at the operating system command level. For example, to run the composition monitoring program, the command CYCLER4 is issued in response to the C> prompt. The executable programs check for the presence of the data files, and attempt to copy the data into memory. If successful, the programs run automatically. If not successful, the programs issue error messages, indicating what data conversion error occurred, and then exit to the operating system command level. In their initial implementation, the programs generate a massive amount of output to the screen. In trying to track down an erroneous entry to a data file, it is useful to redirect the output to a file. This can be done with a redirection command at runtime. For example, the command

CYCLER4 > RUN.DAT, in response to the C> prompt, will cause CYCLER4 to execute and redirect its output to RUN.DAT. Then RUN.DAT can be sent to a printer or examined with a text editor.

Typical Operating Procedure

The user must determine the information that goes into the description files. In the cell description files, the user must decide which cells are active and which are not. The positioner coordinates that lead to maximum system throughput for a given cell can be determined experimentally with RUNXKEYA.C. From examination of representative spectra collected with the system, the user decides which wavelengths are to be used in the analysis. The sequence of the components does not matter as long as it is kept consistent between the cell description and spectrum description files.

Each active cell that is not a single component calibration standard must be calibrated and verified. This is done by preparing a set of calibration mixtures. The spectrum description file spec0.dat describes the spectra to be taken. Spectrum files must obey the MS-DOS naming rules. Spectra flagged with a 1 will be used for training. Spectra flagged with a 0 will be used for verification. Both

services are provided by CHEMO.FOR. The known composition vector is noted at the time the sample is prepared. The components of the composition vector are listed in the same sequence as the component names in the cell description file. The values are floating point numbers that represent the percentage of each component present. The composition components should total to 100.0.

After the samples are prepared, and the description files are set up, a spectrum of each mixture is collected with SPECTRUM.C. The spectra are saved as ASCII lists of floating point numbers to the files listed in the spectrum description file. After the spectra are collected, the user generates the calibration data from the calibration flagged spectra by invoking CHEMO.FOR.

After the calibration data are generated, the user can also verify its accuracy by CHEMO.FOR. This program reports the predicted compositions and errors of the spectra flagged as verification spectra. If the error is greater than acceptable, there are two possible causes. There may be too much noise in the data, or there may be too much correlation in the samples.

Random noise in the training samples and the verification samples introduces error into the accuracy of the

prediction. [Martens 89, pp. 241-246] If there is too much noise in the system, the user might take the following steps to improve the signal to noise ratio. First, check the laser output power. Realignment and cleaning of the laser mirrors may increase signal output. Second, check that the positioner coordinates really are at the optimal coupling position. Third, check to see if bubbles are forming at the probe tip. Fourth, check the alignment of the interferometer. Finally, add more liquid nitrogen to the detector. Once the signal to noise ratio is brought up to an acceptable level, the training and verification need to be rerun.

To obtain accurate analysis data, the data used to train the system must have a high degree of linear independence. [Martens 89, pp. 67-69] In practical terms, this means that the concentrations of the components in the training samples must have a low level of correlation. If the signal to noise performance of the system is adequate, but the prediction error is unacceptably high, it is possible that the concentrations in the training samples are too highly correlated. The correlation of concentrations should be calculated (perhaps with a statistical analysis program), and a new training set with lower correlations should be formulated. Once the new training set is made up, the training and verification must be rerun.

Only after an accurate analysis matrix has been verified for each active non-calibration cell, should the monitoring program be run. A calibration cell is one that contains a pure concentration of some known strong Raman scatterer such as benzene. Any cell containing a single component is a calibration cell. The normalization flag field in the cell description file of the calibration cell contains the expected value of the maximum peak height. This is taken by observing the spectrum of the calibration sample under nominal operating conditions. If the peak height of the calibration sample drifts over time, a normalization factor is calculated by comparing the actual peak height with the expected peak height. For a given sample cell, in the cell description file, the user may turn off the normalization by entering 0.0 in the normalization flag field, or turn it on by entering 1.0 in the normalization flag field.

Program Maintenance

The software is designed for straightforward maintenance. The source codes are in ASCII, and can be updated with and text editor or word processor that handles ASCII files. Because the programs make extensive use of large matrices, they must use the large memory model.

CYCLER4.C and SPECR0.C must be compiled with a compiler that is compatible with Version 2.01 of the Microsoft Quick C compiler. Due to their large memory demands, they will not compile correctly within the Quick C editing environment. They must be compiled at the operating system command level with the command:

```
C>qcl /c /AL /Gt3 /F 4FFF filename.c
```

qcl invokes the compiler. /c causes the system to compile without linking. /AL invokes the large memory model. /Gt3 causes all data whose size is greater than three bytes to be allocated to a new data segment. /F 4FFF causes the stack size to be set to 4FFF hex bytes. For smaller stack sizes, the programs will compile, but will produce runtime errors due to stack overflow. After compilation, the object code must be linked with the National Instruments object code. Linking is accomplished by issuing the command 'link', without arguments, at the DOS prompt. The linker will query for files. When it asks for object files, the compiled object code and the object file MCIBL.OBJ are listed on one line, separated by spaces. When the linker querys for the library filename, the correct response is llibc7.lib. [Microsoft 88, pp. 67-92] [National 89b, p. 1-1]

Since the chemometric program consists of three different source codes, the parts must be compiled separately, and the object codes linked in a different operation. Compilation at the operating system command level with the command:

```
C>fl /c /Gt filename.for [Microsoft 87, pp. 63,232]
```

fl invokes the compiler. /c causes compilation without linking. /Gt causes all data whose size is greater than 256 bytes to be allocated to a new data segment.

Linking is accomplished by issuing the command 'link', without arguments, at the DOS prompt. The linker will query for files. When it asks for object files, they are all listed on one line, separated by spaces. When it asks for a library file, the correct response is llibfor7.lib.

CHAPTER 9

HARDWARE INTERCONNECTIONS

Wiring

This chapter addresses the hardware interconnections required to set up the system. Specifically, it covers connections from the control computer, to the positioners, the beam dump, and the interferometer.

The Three-Axis Positioners

Each positioner consists of four pieces of hardware plus interconnecting cables. The four hardware items are the plug-in board, the MC5300 controller, the motor driver and the motorized stage. The plug-in board is permanently connected via ribbon cable to the MC5300 controller. The controller is connected through various cables to the motor driver and the translating stage. It is critical that the systems be correctly wired. That is, the wiring for positioner 1 must not be confused with the wiring for positioner 0. The wiring for a given axis must be connected to the motor for that axis.

The stages are oriented and tagged such that the axes correspond to a right hand Cartesian coordinate system. Axis 1 corresponds to the X-direction. Axis 2 corresponds to the Y-direction. Axis 3 corresponds to the Z-direction.

The position encoder for a given axis is the grey rectangle on the stage for the given direction. The encoder cable is permanently connected to the encoder, with a D-shell connector on its far end. The D-shell connector is connected to the encoder connector of the MC5300 Controller for the given axis.

The limit switches for a given axis are in the black rectangular box on the opposite side of the stage from the encoder. The end of the limit switch box has a circular cable connector. The limit switch cable has a circular connector on the stage end, and a D-shell connector on the controller end. The limit switches are connected to the auxiliary connector of the MC5300 controller for the given axis.

The positioner motor is the grey cylinder on the end of the stage. The motor cable is about a foot long, permanently mounted to the motor, and terminated in a circular connector. The motor is connected to the motor driver through a cable with a circular connector at both ends. One

end connects to the motor cable. The other end connects to the "motor" connector on the motor driver. Each axis of the motor driver is connected to the MC5300 controller through a cable with a D-shell connector at both ends. One end is connected to an "Indexer" connector on the Motor Driver. The other end is connected to the corresponding "Motor Driver" connector on the Controller.

The plug-in board plugs into any available long expansion slot in an IBM-AT compatible. The address switches are set such that positioner 0 is at I/O port address 300 hex, and positioner 1 is at I/O port address 310 hex. Each board controls three dimensions of movement.

No external DC power supplies are needed. Each MC5300 controller and each motor driver require 117 VAC power. The motors get their power from the motor driver. When the motor driver is switched off, the motors can be hand turned. When the motor driver is switched on, the motors remain fixed in position and resist hand turning.

Beam Dump

The purpose of the beam dump is to protect the end of the excitation fiber connector during positioning operations. During normal operation of the system, the laser is never

turned off. Before a positioning operation occurs, the beam dump is moved between the laser and the microscope objective to block the laser beam. Once reposiitioning is finished, the beam dump is moved out of the way, and the laser power is coupled into the newly positioned fiber.

The beam dump is moved by a 24 volt DC solenoid, the power to which is controlled by a solid state relay. When a logic low (0 volts) is fed into the relay, the solenoid is energized. When a logic high is fed into the relay, the solenoid is de-energized. Logic signals come from line 2 of the parallel printer port of the computer. The relay and the printer port are both TTL compatible. [IBM 84b]

When the solenoid is energized, it lifts the beam dump out of the path of the excitation laser beam. When the solenoid is de-energized, the beam dump falls into position by the force of gravity.

Interferometer

The control computer is connected to the interferometer through an IEEE-488 bus. A National Instruments IEEE-488 cable is used to connect the IEEE-488 interface card in the control computer to the IEEE-488 interface connector on the

front of the interferometer controller. The connectors are stackable.

In operation of the IEEE-488 bus, both the IBM-AT compatible and the Perkin-Elmer 7500 computer (the system that normally interacts with the interferometer) act as controllers. This creates a conflict. Therefore, one or the other must be physically disconnected from the IEEE-488 bus for the other to operate normally.

The address switch on the front if the interferometer controller is normally set at address 2. This is the address for which all control software will look. It is not to be changed.

A useful technique for troubleshooting the IEEE-488 bus is to use a bus analyzer. A National Instruments GPIB410 bus analyzer card is installed in an outboard microcomputer and connected to the bus through an IEEE-488 cable. It can force logic levels onto the bus, or it can monitor and record all activity on the bus.

Chapter 10

CONCLUSIONS

Future Work

This research has raised several questions for further investigation. In particular, can the overall cost of the system be reduced by using an improved probe design? A separate research project, in which predictions of the computer model are being investigated experimentally, is attempting to answer this question. Can the chemometric predictions be improved through the use of neural networks. A project to investigate this question is under way. Finally, can the signal to noise ratio of the present system be improved? Perkin-Elmer has developed a new amplifier with a claimed improvement of 3 dB as compared to the existing design. One of the new amplifiers is on order and will be installed on the existing system. In addition, a higher powered laser than the one currently in use will be installed on the system.

Operation of the system on the isopropanol column is almost certain to lead to changes in the control software. As operating contingencies are discovered and recovery strategies are devised, these must be added to the control

system. The present system checks if all the hardware is ready, and if all the calibration data are loaded. If not, the system returns to the operating system. Contingencies that occur while the system is operating cannot be handled so simply. To do so would put the column out of operation.

Accomplishments

This research has led to a number of specific accomplishments in optical design of the system. Maximum efficiency in coupling a laser to a fiber is obtained when using a microscope objective whose numerical aperture is slightly less than that of the fiber. The present system couples 86.15% of the laser power into the fiber as compared to a reflection limited maximum of 87.6%. It was shown both experimentally and mathematically that the Raman scattering detected by the Dow probe occurs at distances up to 90 mm from the probe tip, dictating the use of a large sample cell. A procedure using ray matrices has been developed to design a fiber bundle to instrument coupling system that assures maximum coupling efficiency.

The effect of long fibers on the system has been investigated. The attenuation of the connections averages 0.9 dB. Attenuation of the 35 meter fiber is too small to measure. Silica Raman scattering in the fiber is a much

bigger problem than attenuation. However, it can be overcome by several means.

Software to calibrate and operate the system has been developed and tested. At this point, everything works except the direct automatic transfer of spectra from the interferometer to the control computer.

Table 10-1 indicates the overall success of the research. Nineteen acetic extract spectra were collected through the 35 meter fiber bundle. They were used to train both the CIRCOM chemometrics program on the Perkin Elmer 7500 computer and the CHEMO chemometrics program in the control system software. For three verification samples on both systems, the RMS errors for acid and solvent concentration predictions were within the 2% limit.

Table 10-1. Chemometric Errors Compared

Mixture	Component	Actual Composition	CIRCOM Prediction	CIRCOM Error	CHEMO Prediction	CHEMO Error
saw3	Acid	35.468	36.834	-1.366	37.85	-2.382
	Solvent	54.468	54.368	0.1	56.292	-1.824
saw11	Acid	38.296	38.339	-0.043	39.589	-1.293
	Solvent	59.131	57.086	2.045	57.311	1.82
saw15	Acid	55.547	55.774	-0.227	56.457	-0.91
	Solvent	39.539	38.246	1.293	40.017	-0.478
RMS				1.13894		1.583336

LIST OF REFERENCES

[Angel 90] "Wavelength selection for fiber optic Raman Spectroscopy. Part 1," Angel and Myrick, Applied Optics, Volume 29, Number 9, 1990, pp.1350-1352.

[Archibald 88a] "Remote Near-IR Reflectance Measurements with the Use of a Pair of Optical Fibers and a Fourier Transform Spectrometer," Archibald, Miller, Lin and Honigs, Applied Spectroscopy, Volume 42, Number 8, 1988, pp. 1549-1558.

[Archibald 88b] "Raman Spectroscopy over Optical Fibers with the Use of a Near-IR FT Spectrometer," Archibald, Lin and Honigs, Applied Spectroscopy, Volume 42, Number 8, 1988, pp.1558-1562.

[Bell 75] Robert John Bell, Introduction to Fourier Transform Spectroscopy, Academic Press, Harcourt Brace and Jovanovich.

[Bennett 89] Information on the input properties of the Perkin-Elmer 1700 interferometer were furnished by Bob Bennett of Perkin-Elmer Limited in September 1989.

[Cahn 88] "Multivariate Calibration of Infrared Spectra for Quantitative Analysis Using Designed Experiments," Cahn and Compton, Applied Spectroscopy, Volume 42, Number 5, 1988, pp. 865-872.

[CRC 81] Handbook of Chemistry and Physics, CRC Press, Boca Raton, FL, 62nd edition.

[Chase 89] Per various telephone conversations between M. J. Roberts and Bruce Chase at Dow Chemical, Midland, MI.

[Crookell 90] "Fourier Transform Raman Spectroscopy in the Near-Infrared Region," Crookell, Hendra, Mould and Turner, Journal of Raman Spectroscopy, Volume 21, 1990, pp. 85-90.

[Daedal 88] MC5300 Controller (PC23 Indexer), Compumotor Model PC23 OPerators Manual, 88-007015 Revision Y, Daedal, Harrison City, PA, 1988.

[DeBeers 89] "Properties of Diamond," a vendor's brochure, DeBeers Research Laboratory, P.O. Box 916 Johannesburg 2000, South Africa.

[DOE 87] Program Research and Development Announcement (PRDA No.DE-PR07-8-7ID12691), "Advanced Sensor Research and Development for Cement Kiln, Non-Paper Drying, and Distillation Applications," Idaho National Engineering Laboratory, US Department of Energy, Idaho Operations Office, July 25, 1987.

[DuPont 80] Engineering Guide to Elastomers, DuPont, 1980, publication number E-26276.

[Everall 89] "Signal-to-Noise Considerations in FT-Raman Spectroscopy," Everall and Howard, Applied Spectroscopy, Volume 43, Number 5, 1989, pp. 778-781.

[Fredericks 85a] "Materials Characterization Using Factor Analysis of FT-IR Spectra. Part 1: Results," Fredericks, Lee, Osborn, and Swinkels, Applied Spectroscopy, Volume 39, Number 2, 1985, pp. 303-310.

[Fredericks 85b] "Materials Characterization Using Factor Analysis of FT-IR Spectra. Part 2: Mathematical and Statistical Considerations," Fredericks, Lee, Osborn, and Swinkels, Applied Spectroscopy, Volume 39, Number 2, 1985, pp. 303-310.

[Fredericks 85c] "Rapid Characterization of Iron Ore by Fourier Transform Spectrometry," Fredericks, Osborn, and Swinkels, Analytical Chemistry, Volume 57, 1985, pp. 1947-1950.

[Garrison 89] Final Report, "On-Line Chemical Composition Analyzer Development," Arlene Garrison, Emil Muly, Michael Roberts, and Charles Moore, July 1989, A Report to the Department of Energy, Contract No. DE-FC07-88ID12691.

[Garrison 90] "On-Line Chemical Analysis Applied to Distillation Control - A Status Report," Garrison, Trimble, Muly, Roberts, and Kercel, American Laboratory, Volume 22, Number 3, February 1990, pp. 19-27.

[Geladi 86] "Partial Least-Squares Regression: A Tutorial," Geladi and Kowalski, Analytic Chemica Acta, 185 (1986) 1-17, Elsevier Science Publishers B.V., Amsterdam, The Netherlands.

[Greenleaf 89a] CT-6055X SPA/SPG User's Manual, Greenleaf Computers, Inc., 1989

[Greenleaf 89b] VLSI 12 mHz Zero-wait '286 Turbo Mainboard, Greenleaf Computers, inc., 1989

[Griffiths 77] "Interferometers vs Monochrometers: Separating the Optical and Digital Advantages," Griffiths, Sloane and Hannah, Applied Spectroscopy, Volume 31, Number 6, 1977, pp. 485-496.

[Harrison 89] "Trouble Shooting Distillation Columns," Harrison and France, Chemical Engineering, May 1989, pp. 126-133.

[Hecht 87] Eugene Hecht, Optics, Second Edition, 1987, Addison Wesley,Reading, MA.

[Hendra 88] "Use of Optical Fibres in Raman Spectroscopy," Hendra, Ellis and Cutler, Journal of Raman Spectroscopy, Volume 19, 1988, pp. 413-418.

[Honigs 84] "Number of Samples and Wavelengths Required for the Training Set in Near-Infrared Reflectance Spectroscopy," Honigs, Hieftje, and Hirschfeld, Applied Spectroscopy, Volume 38, Number 6, 1984, pp. 844-847.

[IBM 84a] IBM Technical Reference: Options and Adapters, IBM Personal Computer Hardware Reference Library,, part number 6280133, IBM, Boca Raton, FL, August 2, 1984, "Enhanced Graphics Adapter."

[IBM 84b] IBM Technical Reference: Options and Adapters Volume 2, IBM Personal Computer Hardware Reference Library,, part number 6322509, IBM, Boca Raton, FL, April, 1984.

[Kercel 90a] "Development of a Fiber-Optic Based Instrument for On-Line Raman Analysis of Process Chemical Composition", Kercel, Garrison, Roberts and Trimble, Proceedings of Sensors Expo West 90, presented at Sensors Expo West 90, Long Beach, CA, March 15, 1990, published by Helmers Publishing Co. Peterborough, NH, paper number 301C.

[Kercel 90b] "Recent Advances in the Development of a Fiber-Optic Based Instrument for On-Line Raman Analysis," Kercel, Garrison and Roberts, to be presented at the SPIE 1990 International Symposium on Optical and Optoelectronic Applied Science and Engineering, San Diego, CA, July 1990, in press.

[Kreyszig 67] Erwin Kreyszig, Advanced Engineering Mathematics, Second Edition, 1967, John Wiley and Sons, (New York).

[Lerner 88] Lerner and Thevenon, The Optics of Spectroscopy, A Tutorial V 2.0, Instruments SA, 1988, part number 2-900-000.

[Lewis 88] "Near-Infrared Fourier Transform Raman Spectroscopy Using Fiber-Optic Assemblies," Lewis, Kalasinski and Levin, Analytical Chemistry, Volume 60, 1988, pp. 2658-2661.

[Lewis 89] "Extending the Vibrational Limits in Near-Infrared Fourier Transform Raman Spectroscopy", Lewis, Kalasinski and Levin, Applied Spectroscopy, Volume 43, Number 1, 1989, pp. 156-159.

[Malinowski 77a] "Theory of Error in Factor Analysis," Malinowski, Analytical Chemistry, Volume 49, Number 4, 1977, pp. 606-612.

[Malinowski 77b] "Determination of the Number of Factors and the Experimental Error in a Data Matrix," Malinowski, Analytical Chemistry, Volume 49, Number 4, 1977, pp. 612-617.

[Massart 88] Massart, Vandeginste, Deming, Michotte and Kaufman, Chemometrics: A Textbook, 1988, Elsevier, (Amsterdam).

[Martens 89] Martens and Næs, Mulitvariate Calibration, 1989, John Wiley and Sons, (New York).

[McLachlan 86] "Fiber-Optic Probe for Sensitive Raman Analysis," McLachlan, Jewett, and Evans, United States Patent No. 4,573,761, March 4, 1986.

[Melles Griot 85] Optics Guide 3, Melles Griot Catalog, 1985, Library of Congress card no. 85-60624.

[Microsoft 88] Microsooft QuickC Tool Kit, Version 2.0, The Microsoft Corporation, Redmond, WA, 1988.

[Microsoft 87] Microsoft FORTRAN Optimizing Compiler Users Guide, Version 4.1, The Microsoft Corporation, Redmond, WA, 1987.

[Millman 67] Millman and Halkias, Electronic Devices and Circuits, 1967, McGraw-Hill, (New York).

[Myrick 90a] "Comparison of some fiber optic configurations for measurement of luminescence and Raman Scattering," Myrick, Angel and Desiderio, Applied Optics, Volume 29, Number 9, 1990, pp. 1333-1344.

[Myrick 90b] "Elimination of Background in Fiber-Optic Raman Measurements," Myrick and Angel, Applied Spectroscopy, Volume 44, Number 4, 1990, pp. 565-570.

[National 89a] Getting Started with Your GPIB-PCII/IIA and NI-488 MS-DOS Handler, part number 320149-01, National Instruments, Austin, TX, February, 1989.

[National 89b] NI-488 Function Calls for C Languages, part number 320014-02, National Instruments, Austin, TX, August, 1989.

[Newell 90] Many Telephone conversations between S. W. Kercel and David Newell, Daedal, Inc., April and May, 1990

[Newport 86] Projects in Fiber Applications Handbook, The Newport Corporation, Fountain Valley, CA, 1986.

[Newport 87] The Newport Catalog With Applications, Catalog 100, The Newport Corporation, Fountain Valley, CA, 1987.

[Newport 89] Telephone conversation between S. W. Kercel and a Newport technician, December 12, 1989.

[Parker 77] O-Ring Handbook, Parker Seal Company, Lexington, KY, 1977.

[Parker 88] "Fourier Transform Raman Spectroscopy Using a Bench-Top FT-IR Spectrometer," Parker, Williams, Hendra and Turner, Applied Spectroscopy, Volume 42, Number 5, 1988, pp. 796-799.

[Plaza 86] "Simulation et Optimisation des Capteurs a Fibres Optiques Adjacentes," Plaza,Dao, Jouan, Fevrier and Saisse, Applied Optics, Volume 25, Number 19, 1986, pp. 3448-3454.

[Pemberton 90] "Raman Spectroscopy Using Charge-Coupled Device Detection," Pemberton, Sobocinski, Bryant and Carter, Spectroscopy, Volume 5, Number 2, 1990, pp. 26-36.

[Roberts 89] "Comparison of Three Preamplifier Candidate Systems for Detection of Raman Scattered Light," M. J. Roberts, unpublished paper, August 25, 1989.

[Roberts 90] "Development and Application of a Raman Spectroscopy Based Instrument for Multi-Point Composition Analysis and Control of a Distillation Column", Roberts, Garrison, Kercel, Muly and Moore, Proceedings of the ISA/90 International Conference, New Orleans, LA, 1990, in press.

[Roush 90] Letter from Patricia Roush, Perkin-Elmer, Norwalk, CT, to Arlene Garrison, dated May 14, 1990.

[Scheid 68] Francis Scheid, Schaums Outline of Theory and Problems of Numerical Analysis, 1968, McGraw-Hill, (New York).

[Schwab 86] "Normal and Resonance Raman Spectroelectrochemistry with Fiber Optic Light Collection," Schwab, McCreery and Gamble, Analytical Chemistry, Volume 58, 1986, pp. 2486-2492.

[Seasholtz 89] "Quantitative Analysis of Liquid Fuel Mixtures with the Use of Fourier Transform Near-IR Raman Spectroscopy," Seasholtz, Archibald, Lorber, and Kowalski, Applied Spectroscopy, Volume 43, Number 6, 1989, pp. 1067-1099.

[Sheppard 77] "A Theoretical Discussion of the Comparative Performances of Dispersion and Fourier-Transform Interferometric Spectrometers for the Infrared Region," Sheppard, Greenler and Griffiths, Applied Spectroscopy, Volume 31, Number 5, 1977, pp. 448-450.

[Singular 89] Singular SA-Series Analog Input System User Manual, Edition 1.30, part number 600-SAXXXX-0130, Singular Technology Corp., Anaheim, CA, 1989.

[Sprouse 89] Letter from James F. Sprouse, Sprouse Scientific Systems, Paoli, PA, to Arlene Garrison, dated October 27, 1989.

[Sterling 87] Donald Sterling, Technician's Guide to Fiber Optics, 1987, Delmar Publishing, Inc.

[Tenge 87] "Calibration in the Fiber Optic Region of the Near-Infrared," Tenge, Buchanan and Honigs, Applied Spectroscopy, Volume 41, Number 5, 1987, pp. 779-785.

[Webb 89] "Practical Considerations When Using Fiber Optics with Spectrometers," Matthew J. Webb, Spectroscopy, Volume 4, Number 6, 1989, pp. 26-34.

[Weyer 87] "Remote Sensing Fiber Optic Probe NIR Spectroscopy Coupled with Chemometric Data Treatment," Weyer, Becker and Leach, Applied Spectroscopy, Volume 41, Number 5, 1987, pp. 786-790.

[Williams 90a] "Remote Sampling Using a Fibre-Optic Probe in Fourier Transform Raman Spectroscopy," Williams, Journal of Raman Spectroscopy, Volume 21, 1990, pp. 147-151.

[Williams 90b] "Analysis of Small Samples Using Fourier Transform Raman Spectroscopy," Williams, Journal of Raman Spectroscopy, Volume 21, 1990, pp. 143-146.

[Williamson 89] "Near-Infrared Raman Spectroscopy with a 783-nm Diode Laser and CCD Array Detector," Williamson, Bowling and McCreery, Applied Spectroscopy, Volume 43, Number 3, 1989, pp 372-375.

[Yariv 85] Yariv, Optical Electronics, Holt Rinehart and Winston, New York, Third Edition, 1985.

APPENDICES

APPENDIX A

HARDWARE SPECIFICATION

Mixtures to be Tested

The system that results from this research is to be a general purpose analyzer that can be applied to distillation columns in many industries. It will be suitable for analyzing many different kinds of mixtures. To test the system, two mixtures will be used during phase two.

One mixture (referred to as acetic extract) consists of acetic acid, water and a proprietary solvent used by Tennessee Eastman. This is the mixture found in the first commercial distillation column on which the system will be installed in phase three. Since acetic acid is a weak Raman scatterer, this mixture poses a severe test to the on-line sensor.

The other mixture (referred to as isopropanol) is a mixture of isopropanol and distilled water. This is the mixture used in the UT Chemical Engineering distillation column. The prototype will be installed here before it is installed on the commercial columns. Since the Raman peak is a non-linear function of the isopropanol concentration, this mixture poses a severe test to the chemometric analysis algorithm.

Definition of Satisfactory Performance

The concentration of a component in a sample can vary between 3% and 99%. The on-line system must predict the concentration with an error of not more than 2% RMS. Error is defined as actual % less predicted %. RMS error is the RMS value of all composition predictions made on several different samples. The amount of error is in the same units as the concentrations; it is not percent error in the usual sense.

Excitation

The excitation wavelength is 1.064 microns. For suitable signal to noise performance, a two watt single mode laser is needed. To minimize maintenance, a diode pumped YAG laser is desirable, should one come on the market.

Data Collection Time

Data collection time is 3 minutes. The phase two system is designed to monitor five different points. With three minutes per cycle allowed for each monitoring point, then the system should be able to efficiently monitor columns with a fifteen minute time constant. The on-line system is

to generate a suitable Raman spectrum in 2.2 minutes (16 scans). In the remaining 48 seconds the spectrum must be transformed into a composition analysis, and the control system to reposition the multiplexer optics for the next reading.

Probe

Probes should be of the Dow 6-around-1 design. The probe housing should be between 6 and 8 inches long and made of stainless steel. The probe should be no more than 5/8" diameter. The end of the probe should have a quartz window 0.5 mm to 1.0 mm thick. O-rings used in probe assembly should be made of Kalrez, and must not be vulnerable to attack by the sample or by common solvents such as toluene.

Resolution and Bandwidth

For a general purpose instrument the resolution must be two wave numbers. A minimum bandwidth of 600 wave numbers is needed. For some mixtures, a bandwidth of 3000 wave numbers may be required. To meet these requirements, an interferometer must be used to obtain the spectrum

Multiplexers

Time division multiplexing will be accomplished with computer controlled 3-axis motorized optical translation tables. There will be one table for the excitation fiber and one for the receiving bundle. The tables will have at least 2 inches of movement in each dimension. The maximum translation rate will be on the order of 2 inches per second. A single circuit board which can be plugged directly into an IBM-AT compatible computer will control the three linear positions of a three-axis linear translation table. There will be one limit switch per axis to indicate a home position. There will be one limit switch per axis to indicate the end of travel position. There will be an absolute position encoder that can be read by the controlling circuit board. The encoder will determine absolute position to within at least 0.001 mm resolution and at least 0.005 mm accuracy. The system shall be capable of moving the table along any of the three axes in steps of as small as 0.001 mm.

Chemometric Software

Chemometric analysis will be performed by the control computer. Multiple linear regression will be used in the initial implementation.

APPENDIX B

CONTROL SYSTEM SOFTWARE SPECIFICATION

Hardware

Interfaces between the analyzer control computer (an IBM-AT compatible, UT Tag Number 399509) and the other hardware is shown in Figure B-1. This is the configuration of the system in its operational form on an actual distillation column. Not all these features need to be implemented for laboratory testing of the instrument.

An RS-232 port is the interface with the column process control computer. A composition data stream will be continuously fed to the process control computer.

The analyzer control computer issues commands to, and receives spectra from, the 68000 based interferometer controller via an IEEE-488 (GPIB) bus.

The status of the system will be monitored by detecting various analog voltages. These will be fed into the analyzer control computer via the analog to digital convertor. An analog test point voltage, proportional to the infrared detector signal level will be used for peaking positioners, if needed. The presence (or absence) of scattered signal will be detected by the photodiode. The rate of fluid flow into the sample cells will be detected by analog flow detectors.

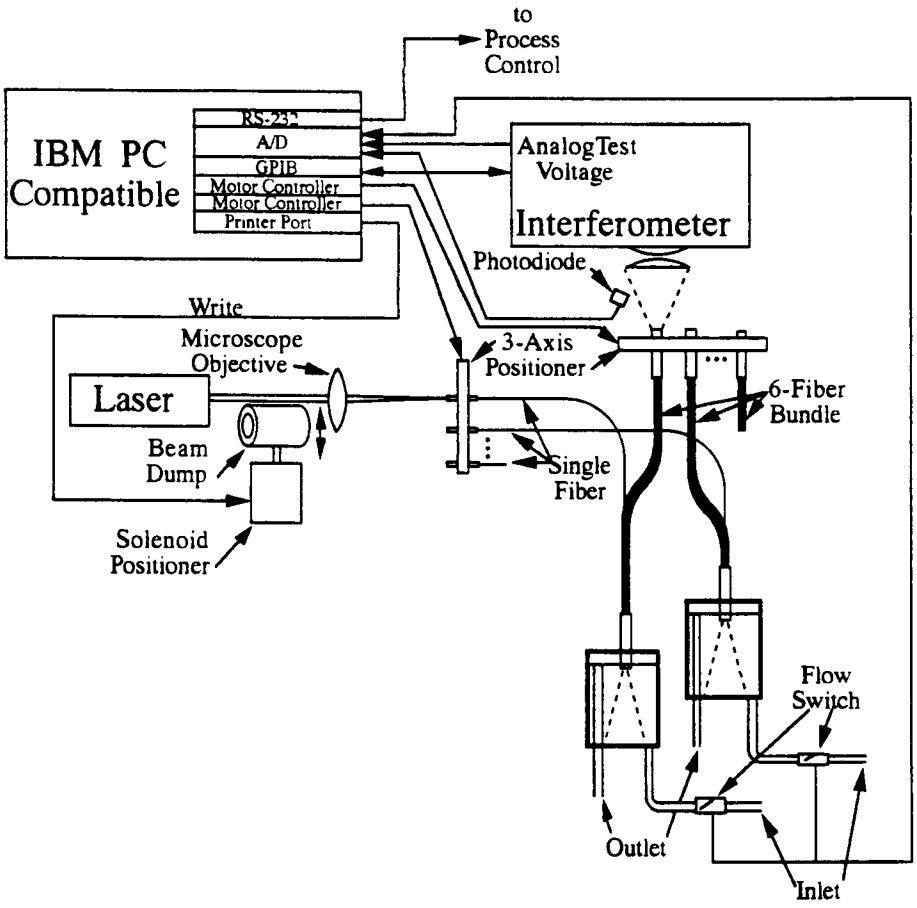


Figure B-1. Control System Interfaces

Optical multiplexing is done by two 3-axis positioners, one for the excitation fiber, and the other for the receiving fiber bundle. When the analyzer control computer selects a cell, it directs the positioners to move that cell's fibers into the foci of the optical couplers. Each positioner has a motor controller card in the PC. The card handles positioning of three motors, reading current position coordinates from encoders, and checking the status of positioner limit switches.

To prevent the laser from damaging the end connectors of the excitation fibers, the beam dump dissipates laser power during positioning operations. The analyzer control computer sends a flag on one of the control lines of the printer port. The flag is detected by TTL logic which used it to actuate a solenoid. The system moves the beam bump into the path of the infrared beam before the positioning operation begins. It moves the beam dump out of the path of the beam before the task of taking the spectrum begins.

Assumptions

In the operational version of the on-line composition sensor, the following conditions are expected:

1. In normal operation, fluid will flow continuously through the sample cell. The flow rate will be fast enough that the entire volume of the cell will be exchanged at least once per column time constant.

2. Rate of flow of fluid into the sample cell will be measured by a sensor. The sensor will return a binary flag to the control software. A TRUE condition means the flow rate is above the safe threshold. A FALSE condition means the flow rate is below the safe threshold and requires operator intervention.

3. Temperature in the sample cell will be measured, but is not expected to be used by the control software.

4. Chemometric analysis of spectra will be done by a multiple linear regression function built into the control software.

5. Infrared spectra can be used for analysis directly. They need not be transformed into Raman offsets.

6. A single IBM-AT compatible will handle both control and analysis.

7. Composition data will be fed to the column control system in a continuous stream via an RS-232 line.

8. For each reading made by the analyzer, the data stream to the column control computer will include the time, date, cell number, number of component, identifying strings and measurements for each component, and an error byte.

9. The error byte will consist of flags with indications of contingencies.

10. Positioners will be set to a remembered set of coordinates. They will not be automatically peaked unless experience with the system on the isopropanol column indicates that it is necessary.

Operating Cycle

In the operational version of the analyzer on the column, for normally unattended operation, the operating cycle program (cycle2.exe) will have the following steps:

1. Select the next connected sample cell.
2. Move the beam dump into the blocking position.
3. Move the selected excitation fiber into the focus of the input coupler.
4. Move the selected receiving fiber bundle into the object area of the output coupler.
5. Move the beam dump to the unblocked position.
6. Check the photodiode to see if scattered light is being generated.
7. Reposition the receiving fiber bundle for peak test point voltage. (Note: This step will only be included if testing indicates that it is necessary.)
8. Issue a command to the interferometer to collect a spectrum.
9. Wait for the interferometer to issue a "spectrum complete" message. Waiting time should have an upper bound.
10. Issue a command to the interferometer to send the spectrum to the analyzer control computer.

11. Convert the spectrum into a format usable by the chemometric function.
12. Compute the composition from the spectrum.
13. Formulate the composition message.
14. Save the composition message to a file.
15. Send the composition message to the process control computer via the RS-232 port.
16. Go to step 1 and repeat the cycle.

Initial laboratory testing will be attended operation, and a simpler operating cycle program (cycle1.exe) will be used:

1. Select the next connected sample cell.
2. Move the beam dump into the blocking position.
3. Move the selected fibers into the foci of the couplers.
4. Move the beam dump to the unblocked position.
5. Issue a command to the interferometer to collect a spectrum.
6. Wait for the interferometer to issue a "spectrum complete" message. Waiting time should have an upper bound.
7. Issue a command to the interferometer to send the spectrum to the analyzer control computer.

8. Convert the spectrum into a format usable by the chemometric function.
9. Compute the composition from the spectrum.
10. Formulate the composition message.
11. Save the composition message to a file.
12. Poll the key buffer. If a Ctrl-X has been entered, close the output file and return to the operating system; otherwise, return to step 1 and repeat the cycle.

Training the Chemometric Algorithm

Two additional programs are needed to collect training spectra and to train and verify the chemometric algorithm. The collection program (spectrum.exe) will consist of the following steps:

1. Read the spectrum and cell description files.
2. Move the beam dump into the blocking position.
3. Move the selected fibers into the foci of the couplers.
4. Move the beam dump to the unblocked position.
5. Wait for the user to indicate that the next sample is ready.
6. Issue a command to the interferometer to collect a spectrum.

7. Wait for the interferometer to issue a "spectrum complete" message. Waiting time should have an upper bound.
8. Issue a command to the interferometer to send the spectrum to the analyzer control computer.
9. Convert the spectrum into a format usable by the calibration program.
10. Query the user whether or not the spectrum is satisfactory. If it is, save it to a file; if it is not, collect a new spectrum for this sample.
11. If the last spectrum saved is the last in the list of spectra to be collected, return to the operating system; otherwise, return to step 5 and collect the next spectrum.

The training program (chemo.exe) will consist of the following steps:

1. Read the spectrum description, cell description, and spectrum data files.
2. Using multiple linear regression, compute the analysis data (this is the actual training step).
3. Calculate composition from verification spectra.
4. Store the regression data to a file.
5. Return to operating system.

External Files

Cell data files (cella0.dat through cella4.dat):

These are ASCII files prepared by the user with a text editor. Each file will contain information about a particular sample cell. It will contain the cell number, an "in use" flag (clear if the cell is not being used), the number of components in the cell mixture, a list of component identifying strings, the number of samples used for training, the name of the file containing the trained matrix for this cell, and stage coordinates for this cell.

Spectrum files:

These are files generated by the collection program and used by the training program. They will contain spectral data in a convenient format.

Matrix files (matrix0.dat through matrix4.dat):

These are files that contain the data generated by the training program. The cycle program will use the data contained in this file, along with the current spectrum to compute composition.

Spectrum description file (spec0.dat):

This ASCII file contains the cell number that is to be calibrated, the count of spectra, a list of spectrum files, and a list of corresponding composition vectors.

Output file (messages.dat):

This ASCII file is a running log of the composition messages generated during the operating cycle.

Output Data

During the operating cycle, each composition reading will generate an ASCII string containing the composition data. Each datum will be delimited by a space. The string will include the time, date, cell number, error byte, peak spectrum value, number of components, a list of component pairs (each pair being a string and a composition number), and a string ending delimiter of "AR."

All inferences (such as "broken fiber") will be drawn by the process control computer, based on the contents of the error byte. The contingencies that will use the bit positions in the error byte will be determined by operating experience in the latter part of phase 2.

APPENDIX C

EXPERIMENTAL DATA: REPRESENTATIVE SPECTRA

This Appendix includes Raman spectra for three representative distillation mixtures. The Eastman mixture consists of acetic acid, water, and a proprietary entraining solvent. The UT Chemical Engineering mixture consists of isopropanol and water. The Hercules mixture consists of cumene hydroperoxide, dimethyl benzyl alcohol, acetophenone, and cumene. In addition, spectra for benzene are included. Benzene is a strong Raman scatterer and is used as a calibration standard.

Free optics spectra were taken using the sample chamber in the conventional configuration if the FT-Raman system. Short fiber spectra were taken both with the Dow probe (6-around-1 design) and the EIC probe (4-around-1 design). Long fiber spectra were taken with a Dow probe, but using low water fibers. The fiber bundle is 35 meters long for both excite and receive.

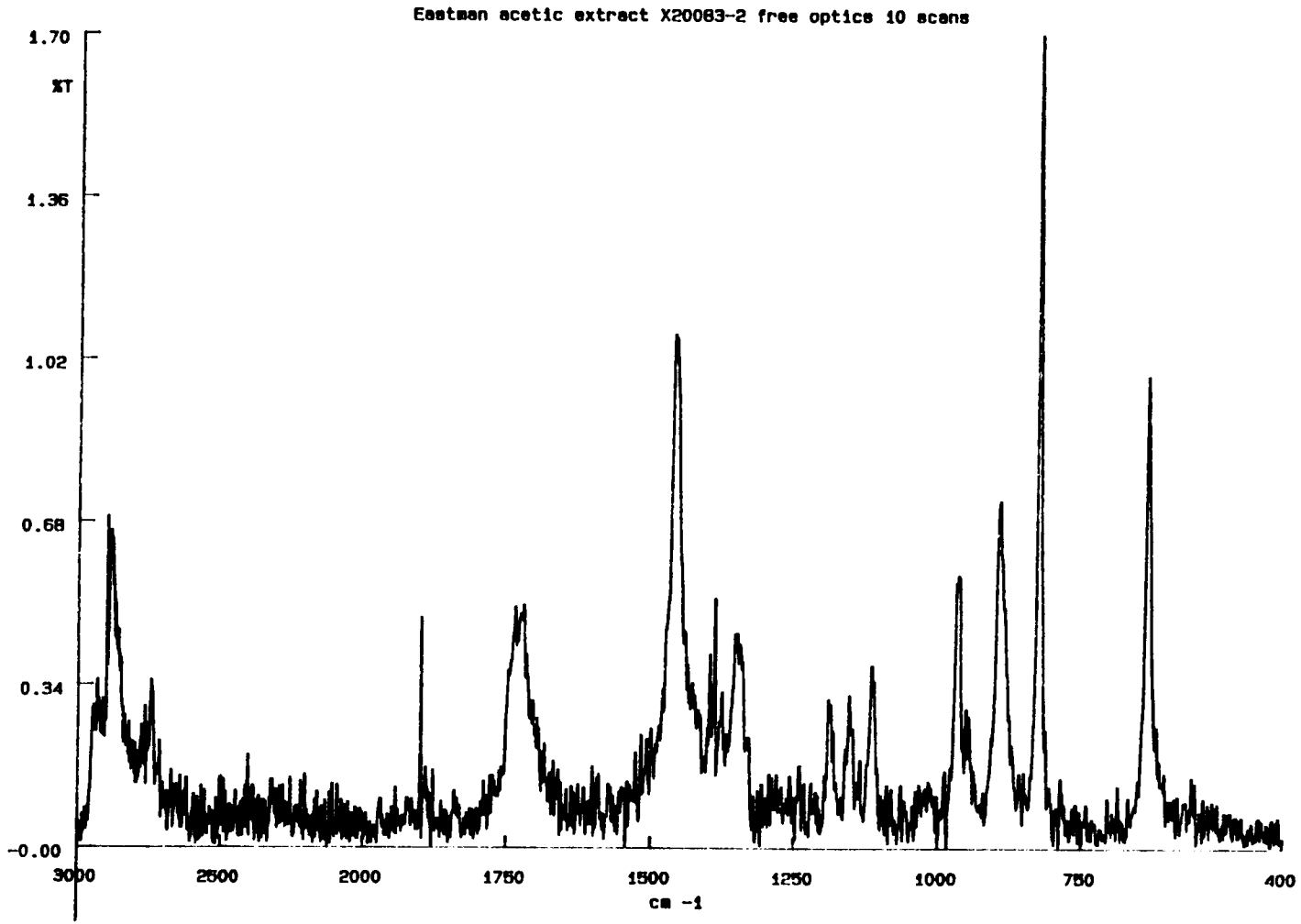


Figure C-1. Acetic Extract - Free Optics

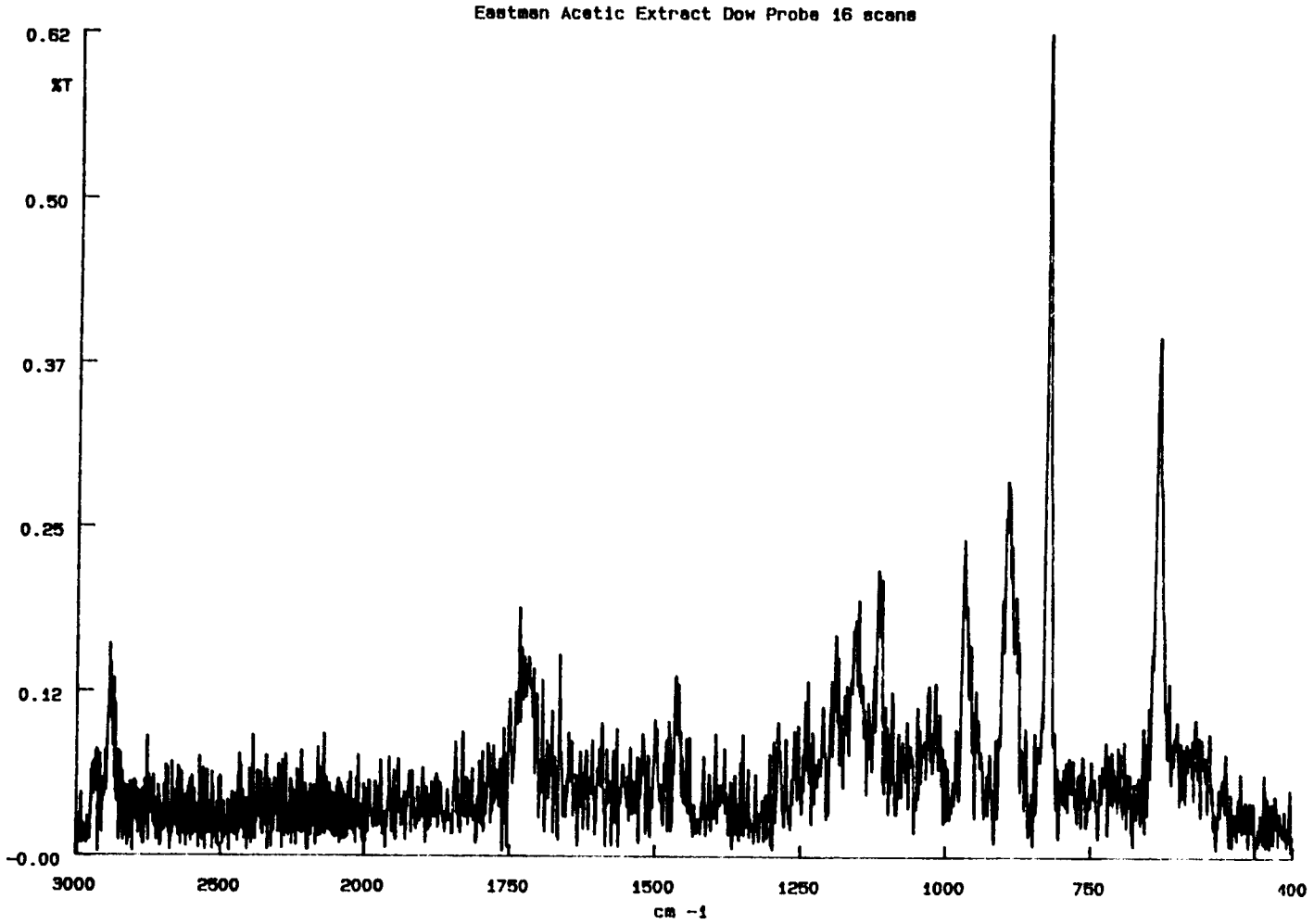


Figure C-2. Acetic Extract - Dow Probe Short Fiber

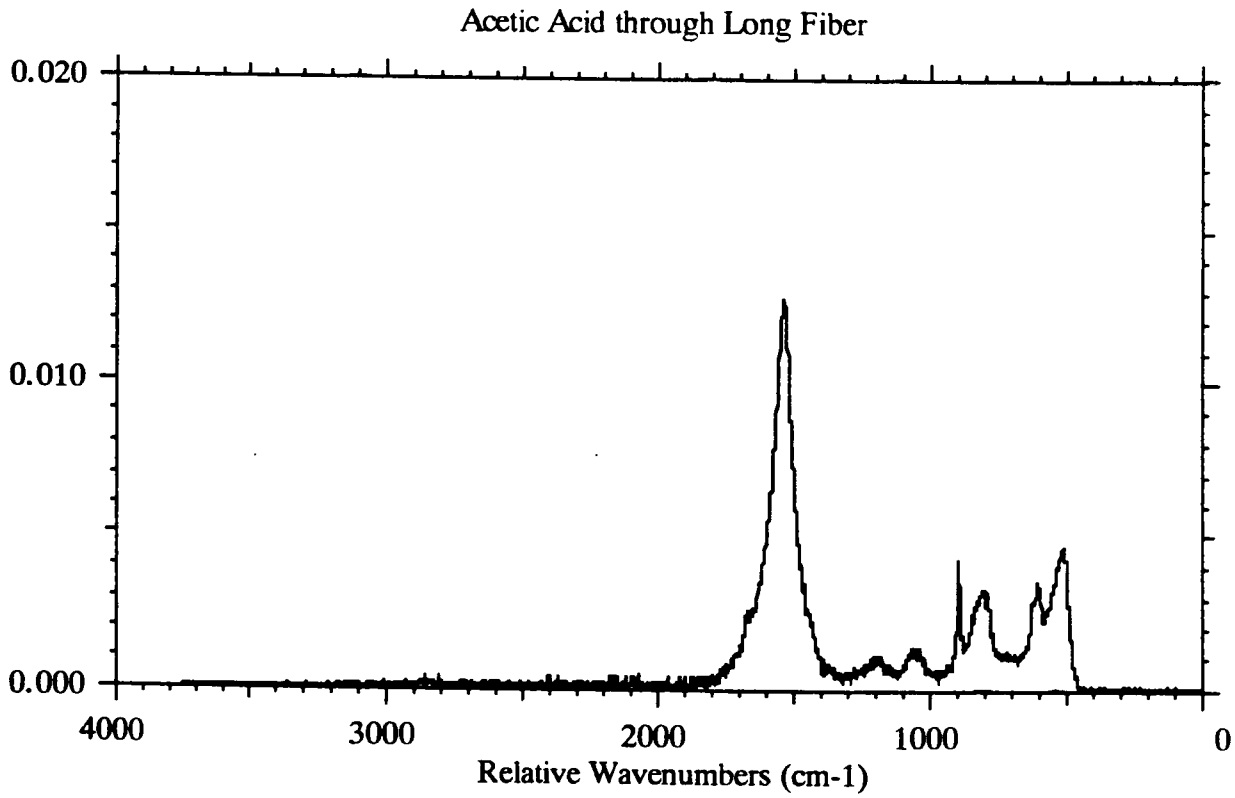


Figure C-3. Acetic Extract - Dow Probe Long Fiber

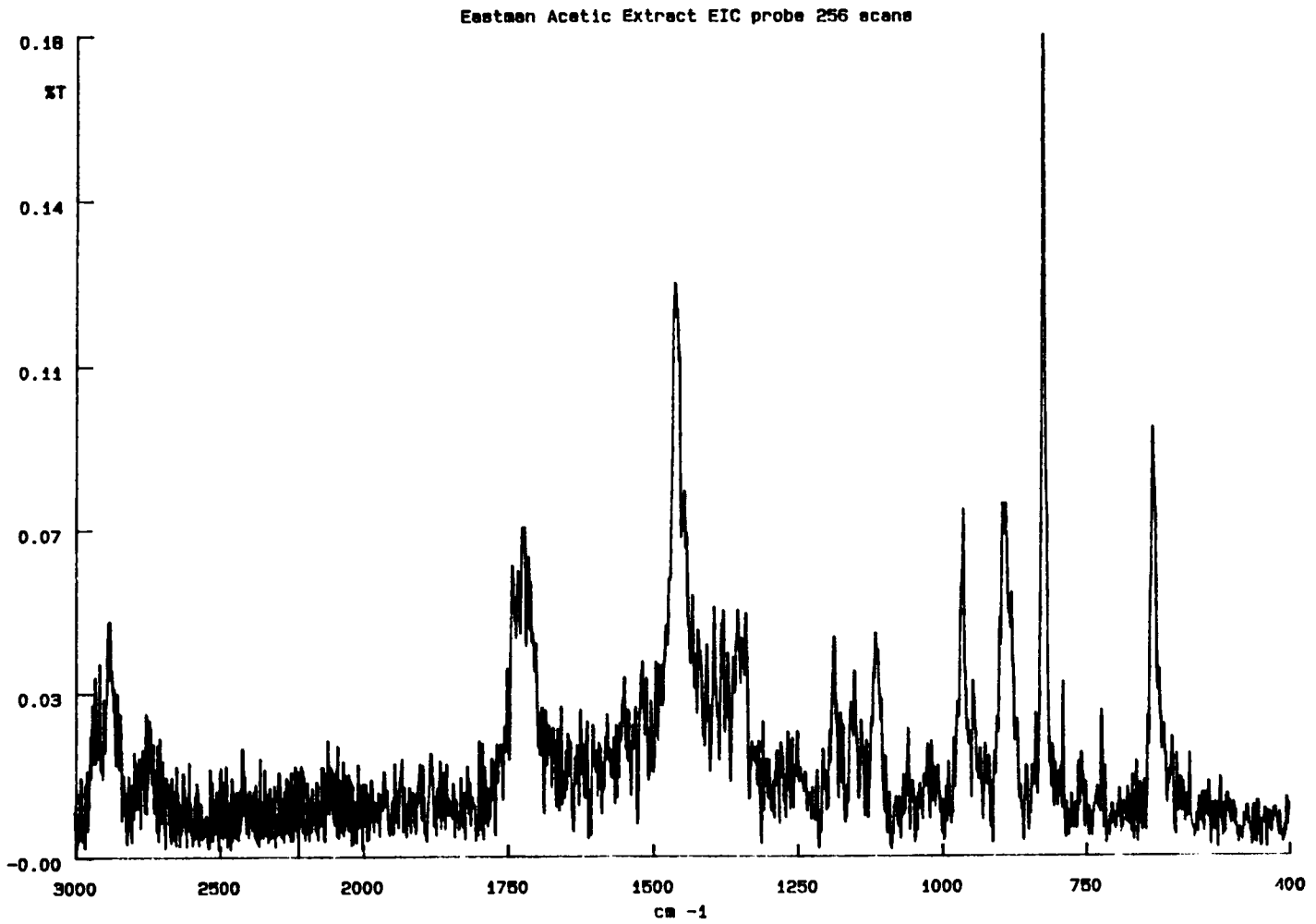


Figure C-4. Acetic Extract - EIC Probe

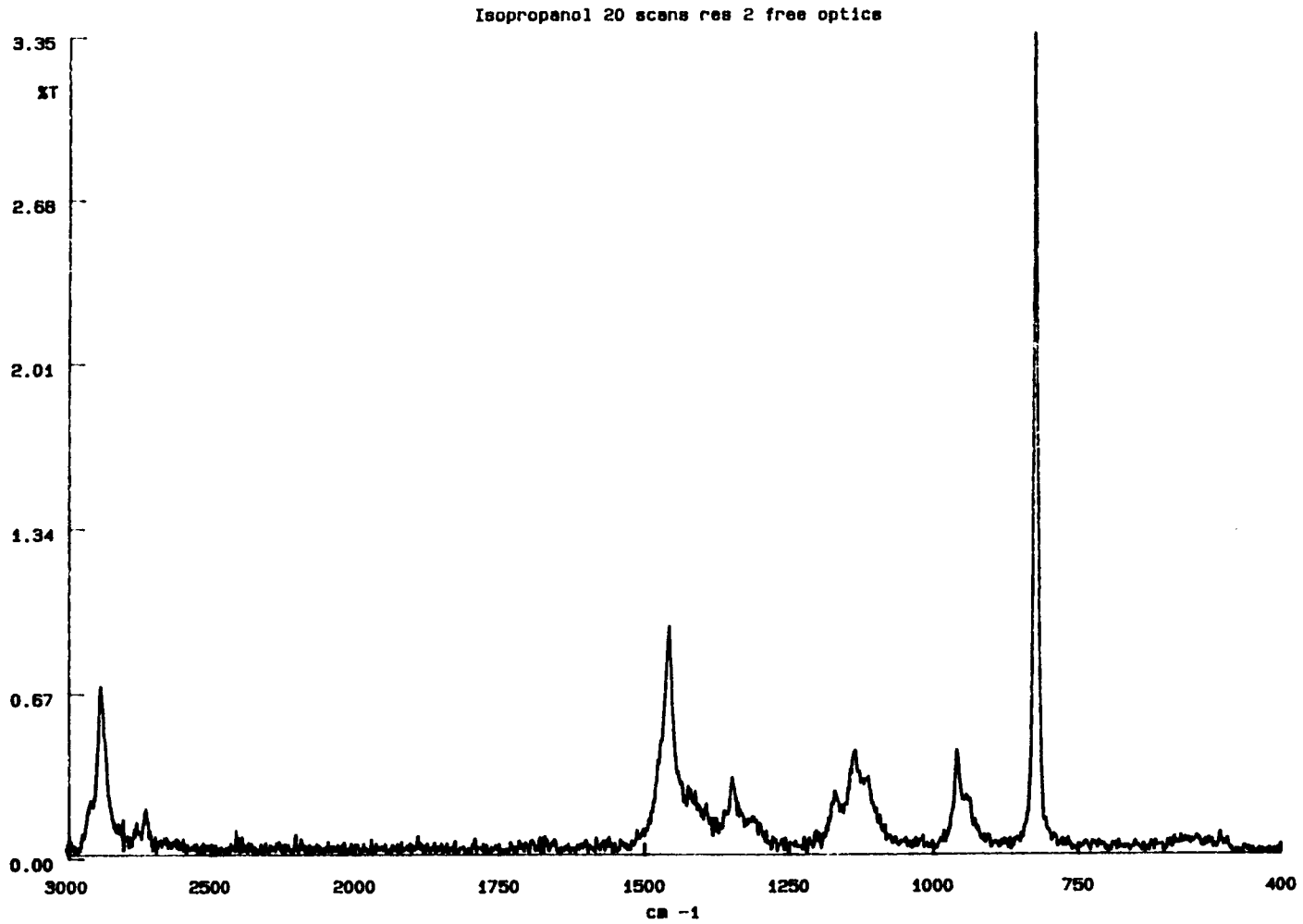


Figure C-5. Isopropanol - Free Optics

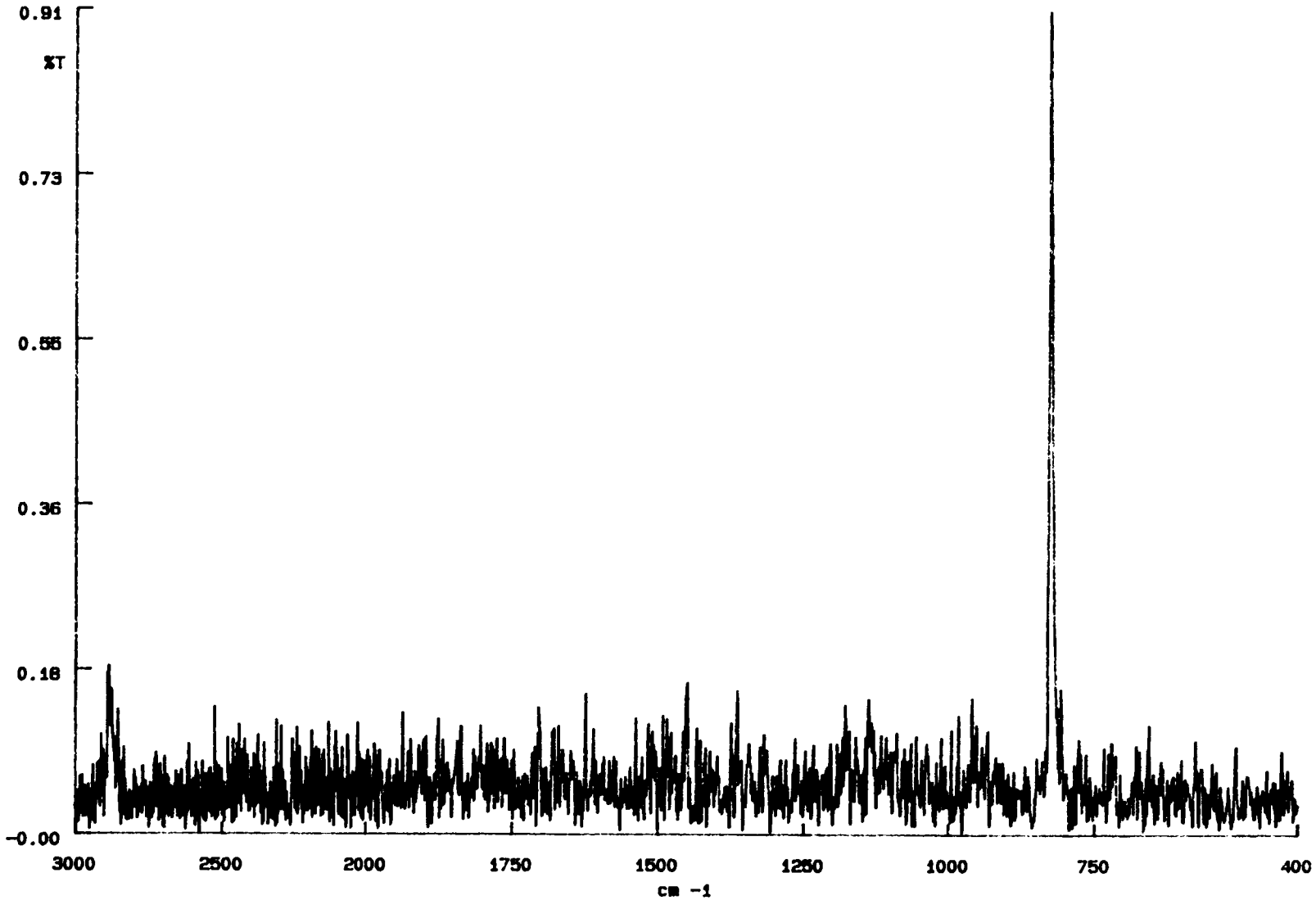


Figure C-6. Isopropanol - Dow Probe Short Fiber

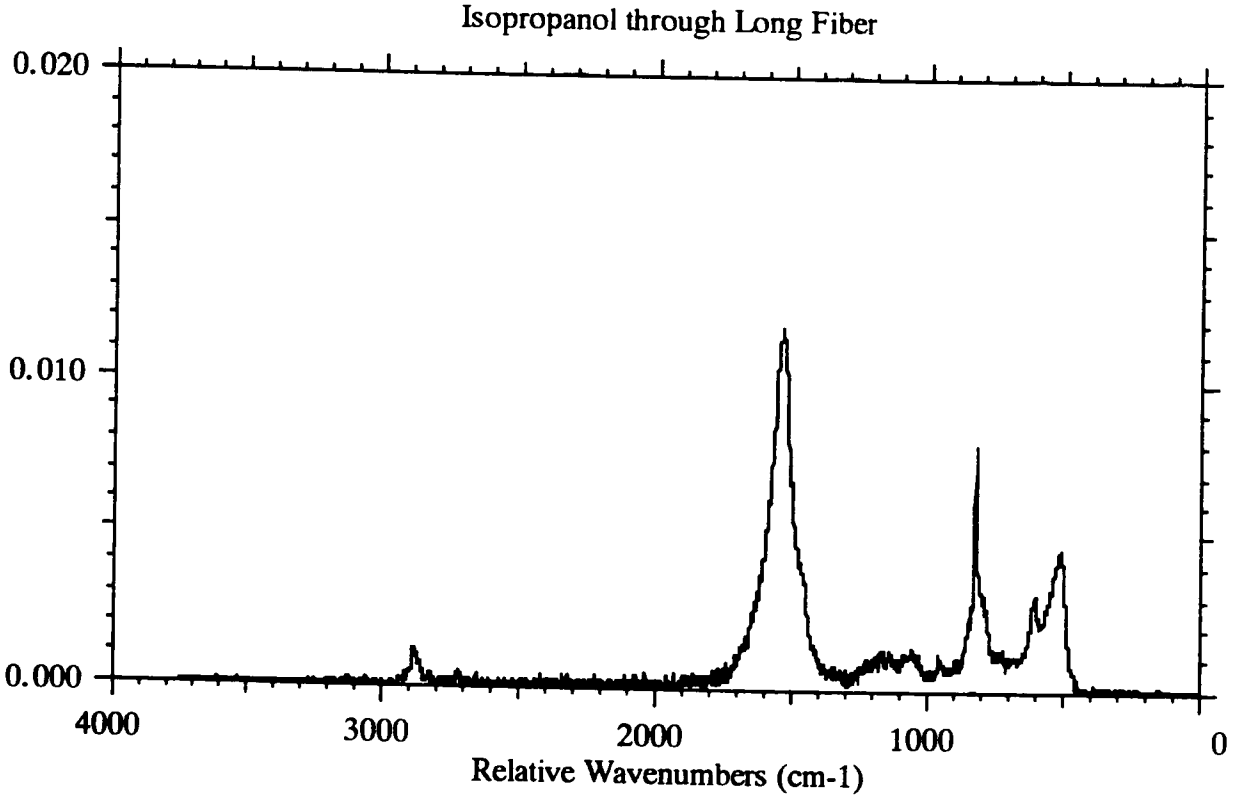


Figure C-7. Isopropanol - Dow Probe Long Fiber

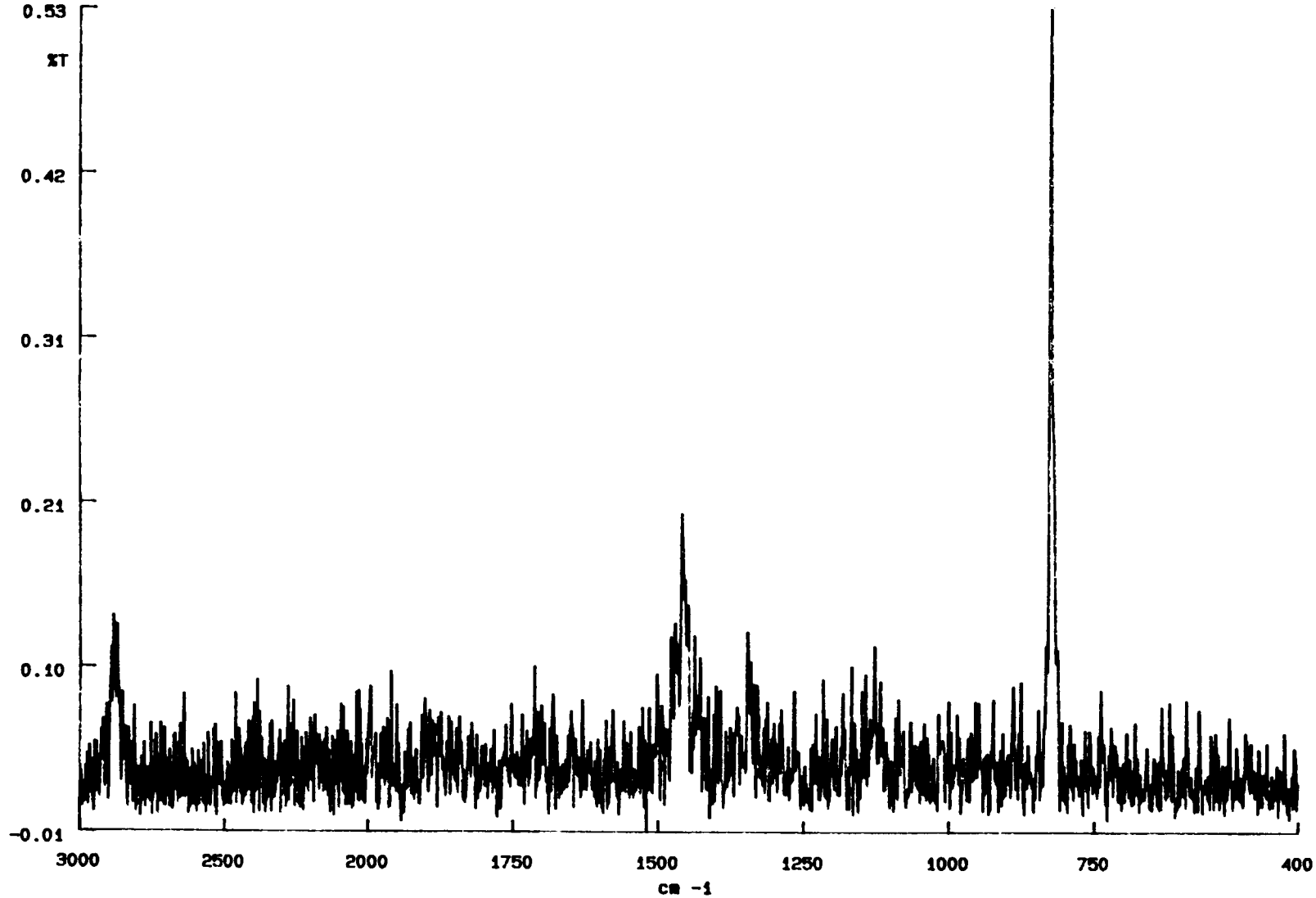


Figure C-8. Isopropanol - EIC Probe

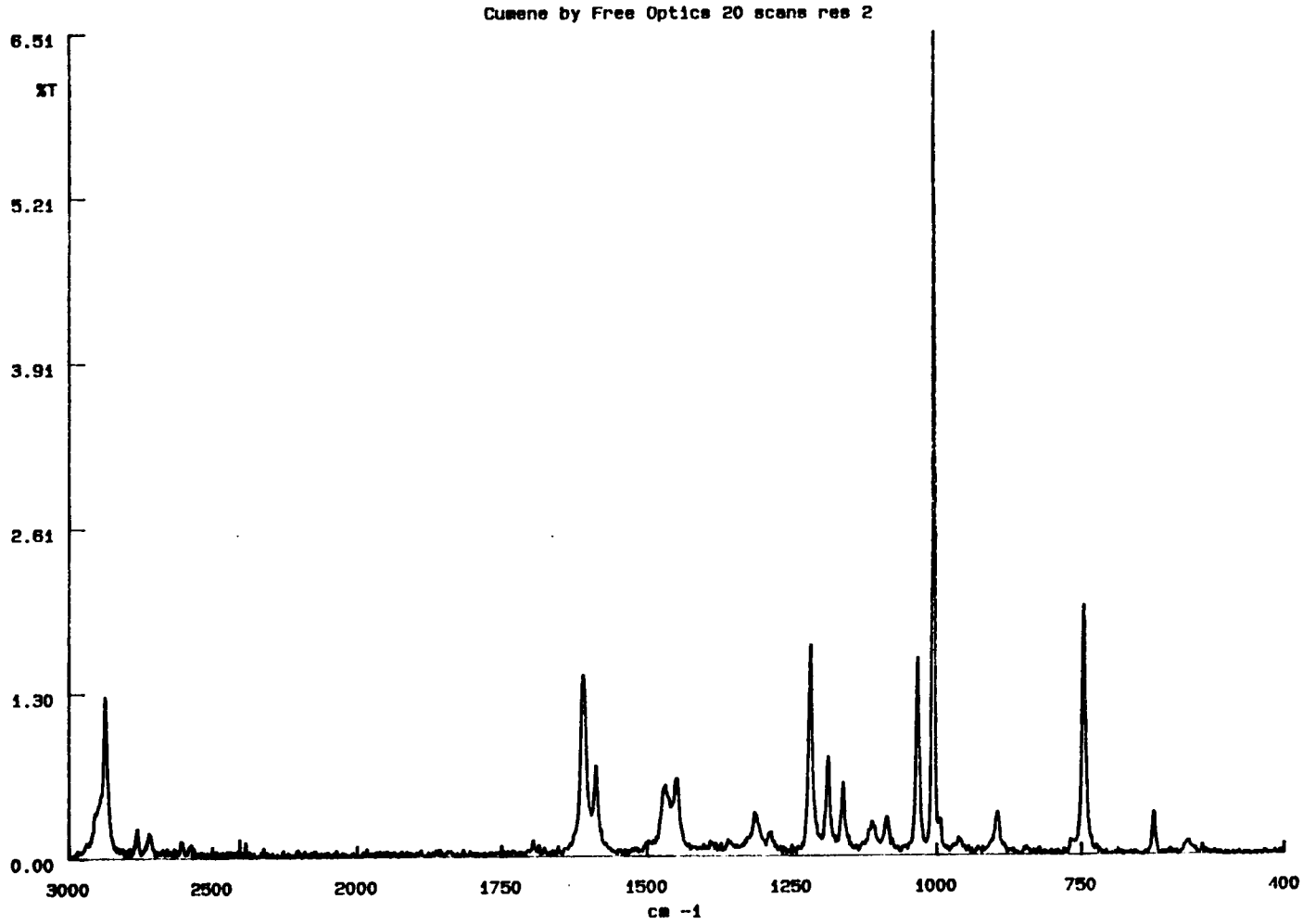


Figure C-9. Cumene - Free Optics

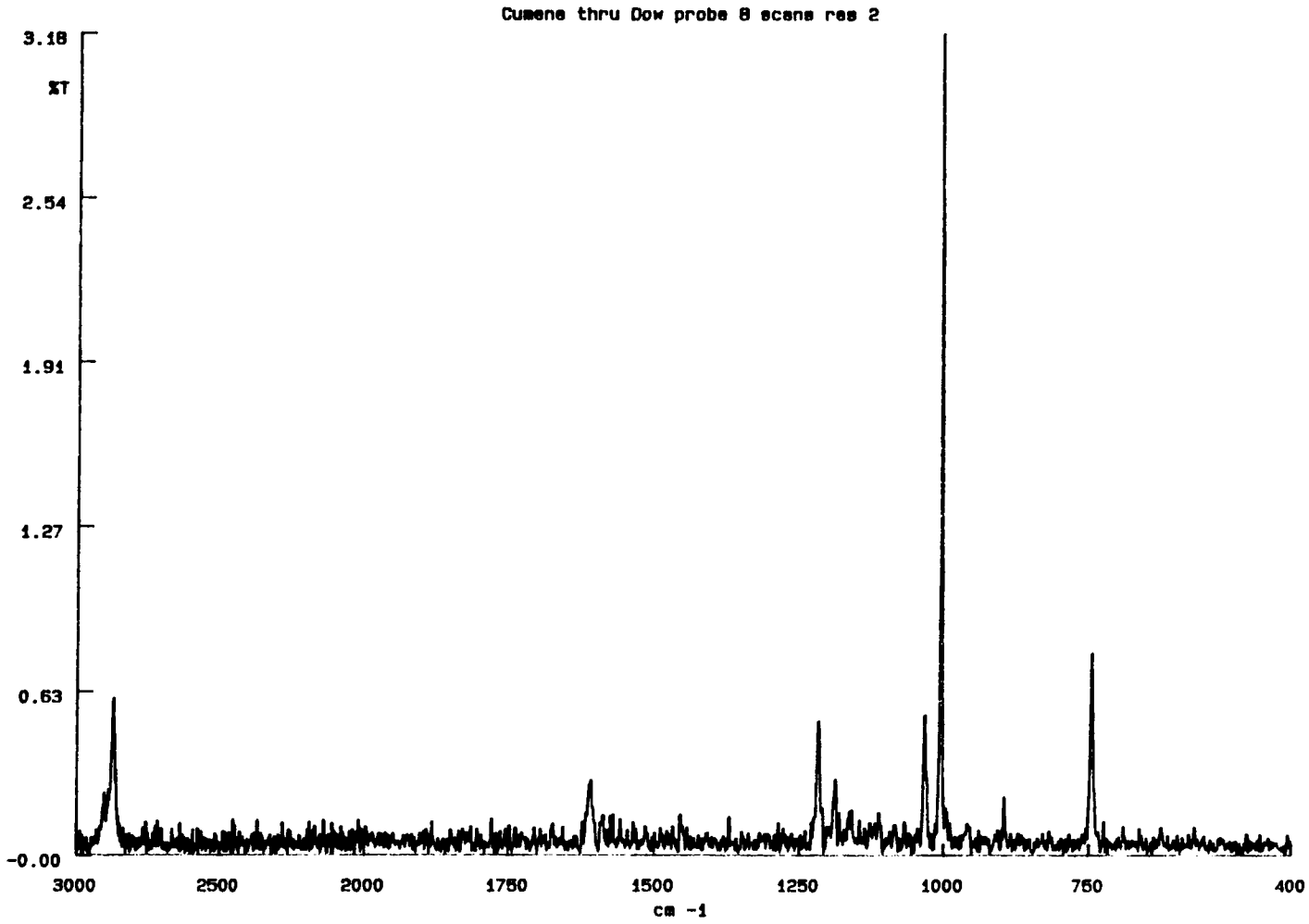


Figure C-10. Cumene - Dow Probe Short Fiber

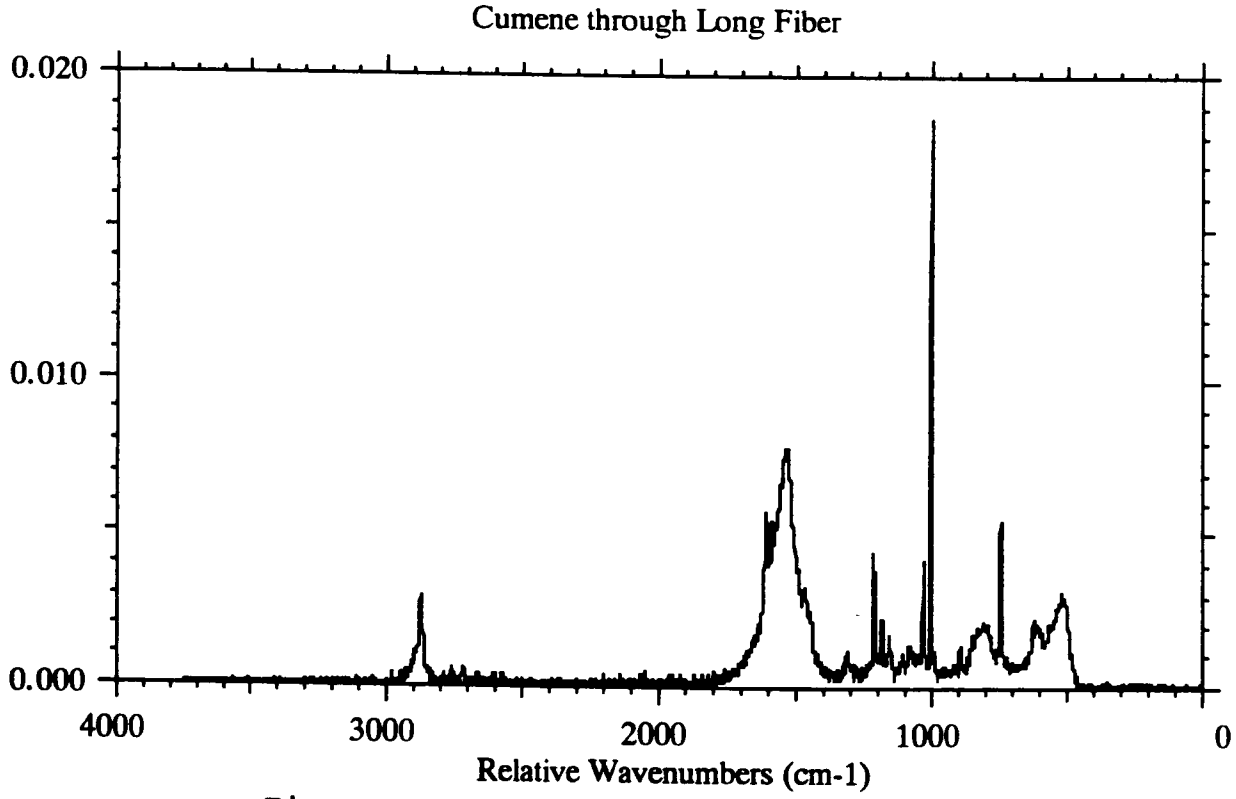


Figure C-11. Cumene - Dow Probe Long Fiber

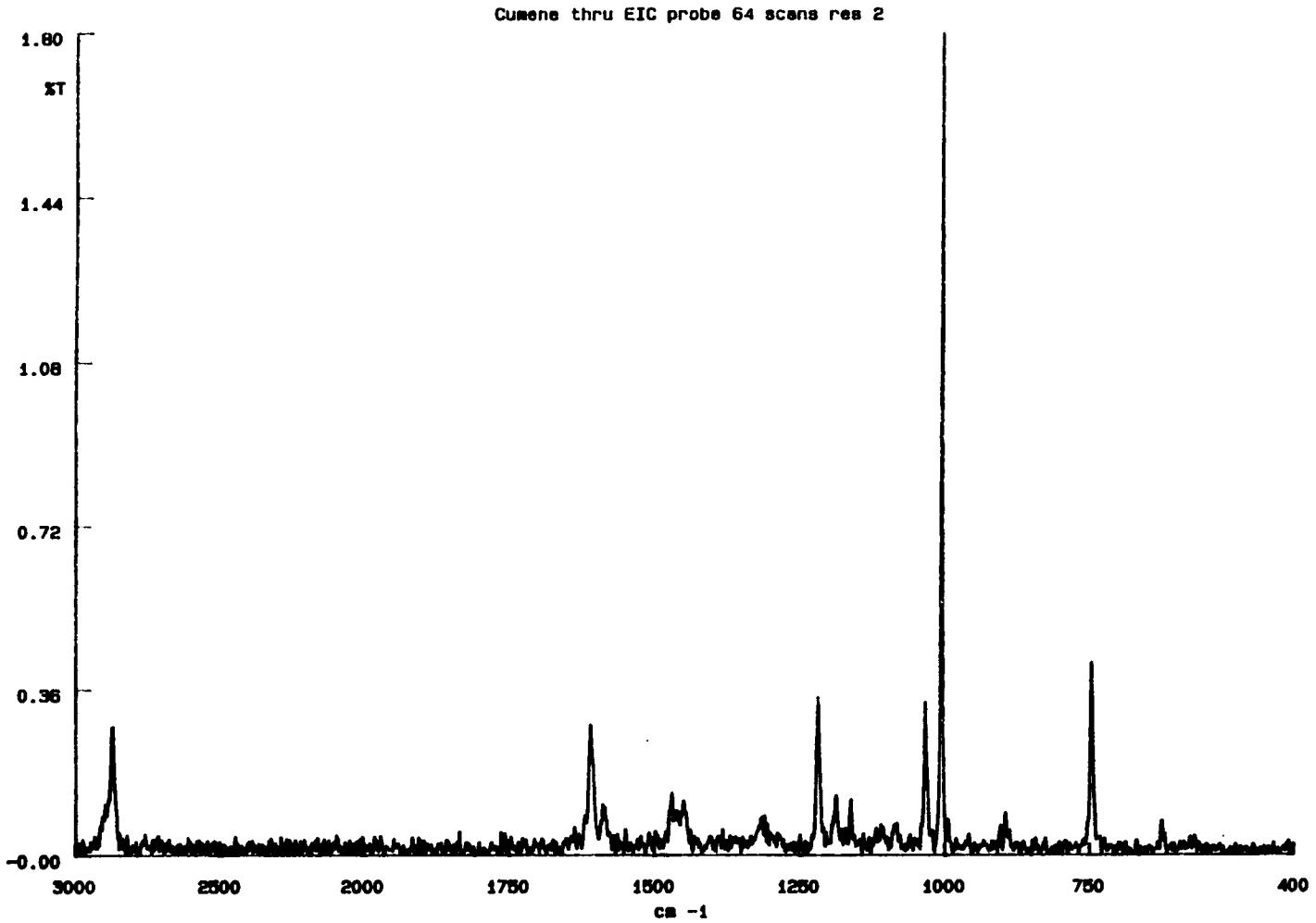


Figure C-12. Cumene - EIC Probe

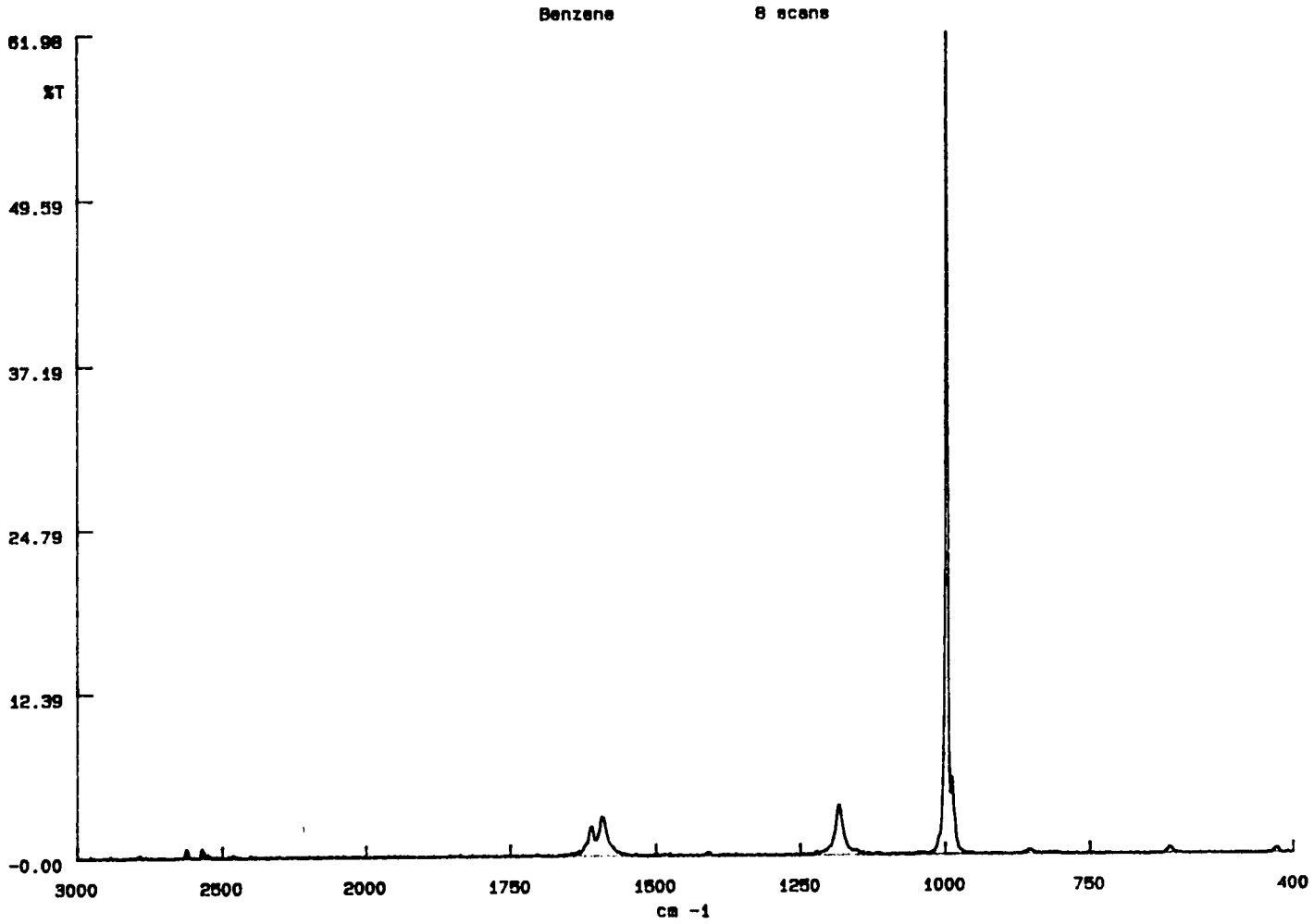


Figure C-13. Benzene - Free Optics

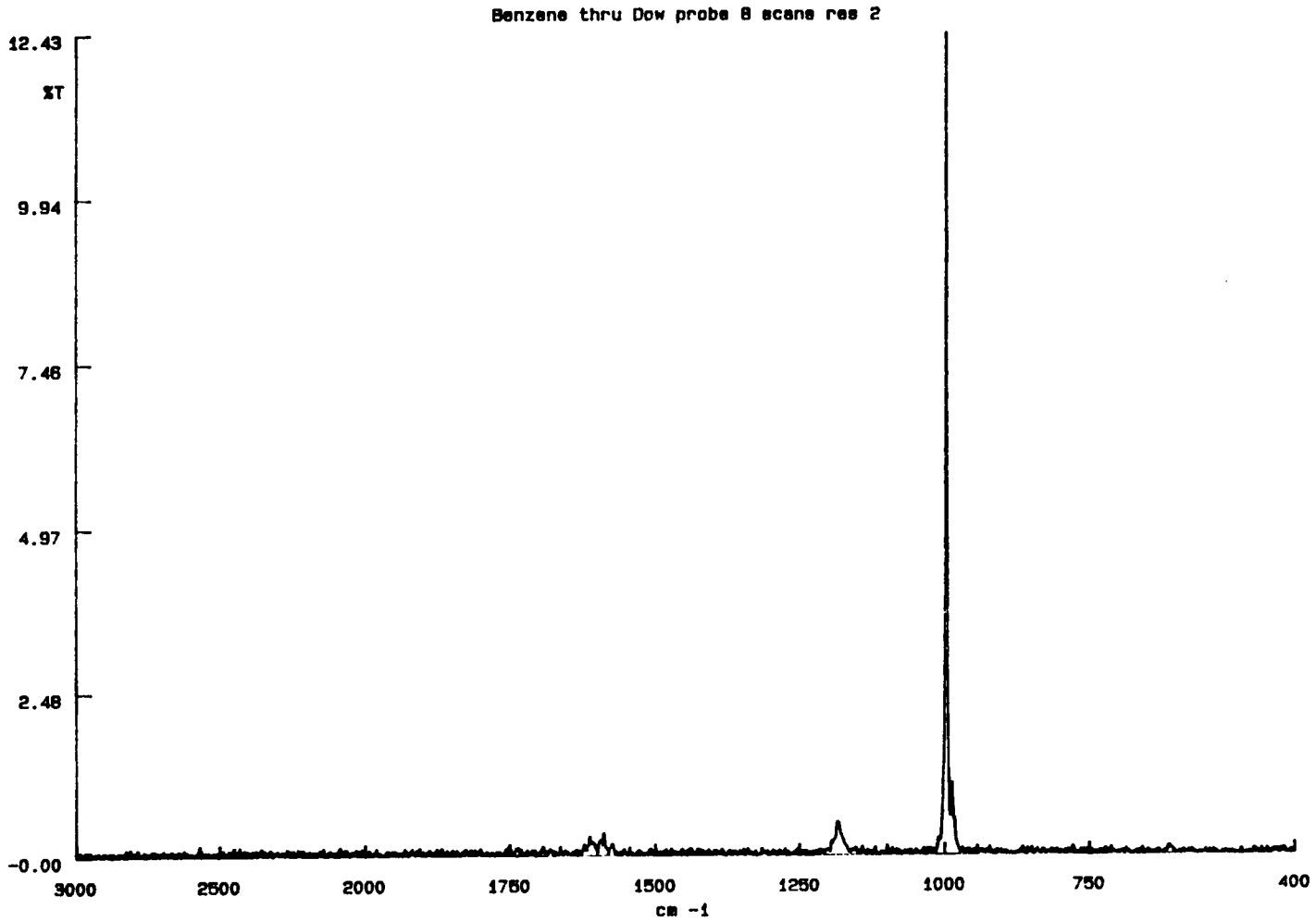


Figure C-14. Benzene - Dow Probe Short Fiber

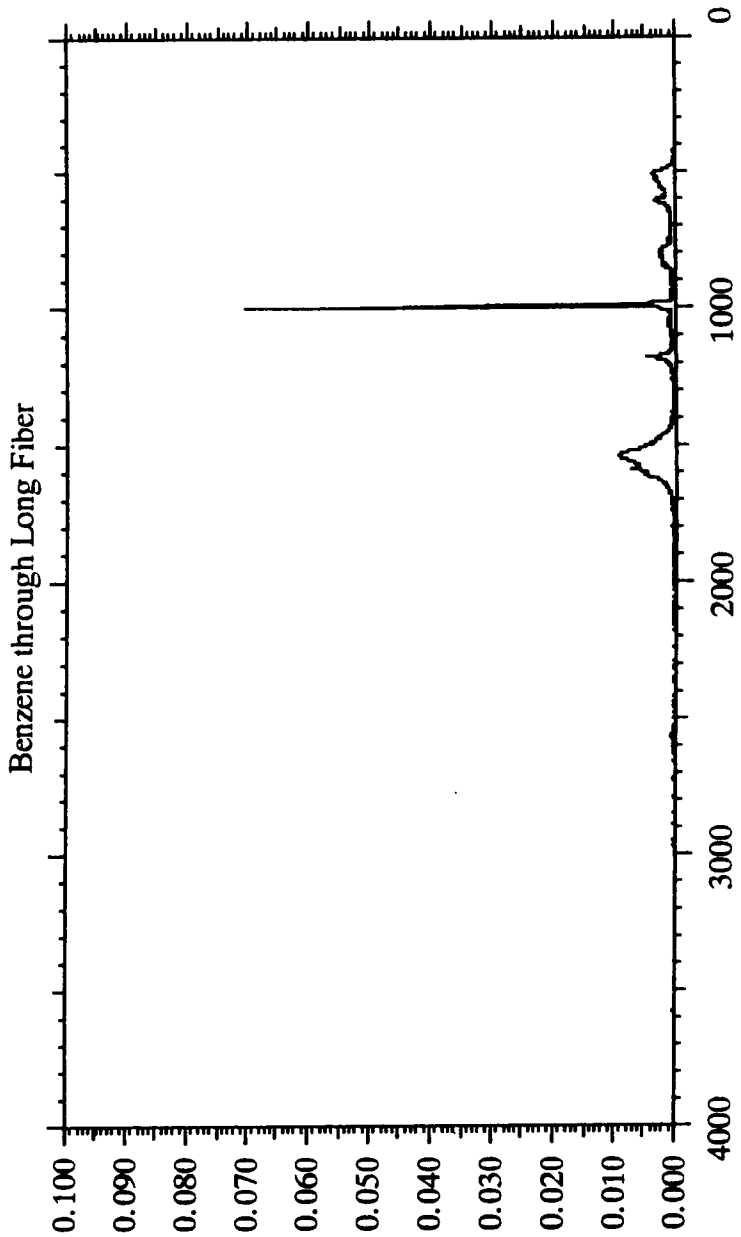


Figure C-15. Benzene - Dow Probe Long Fiber

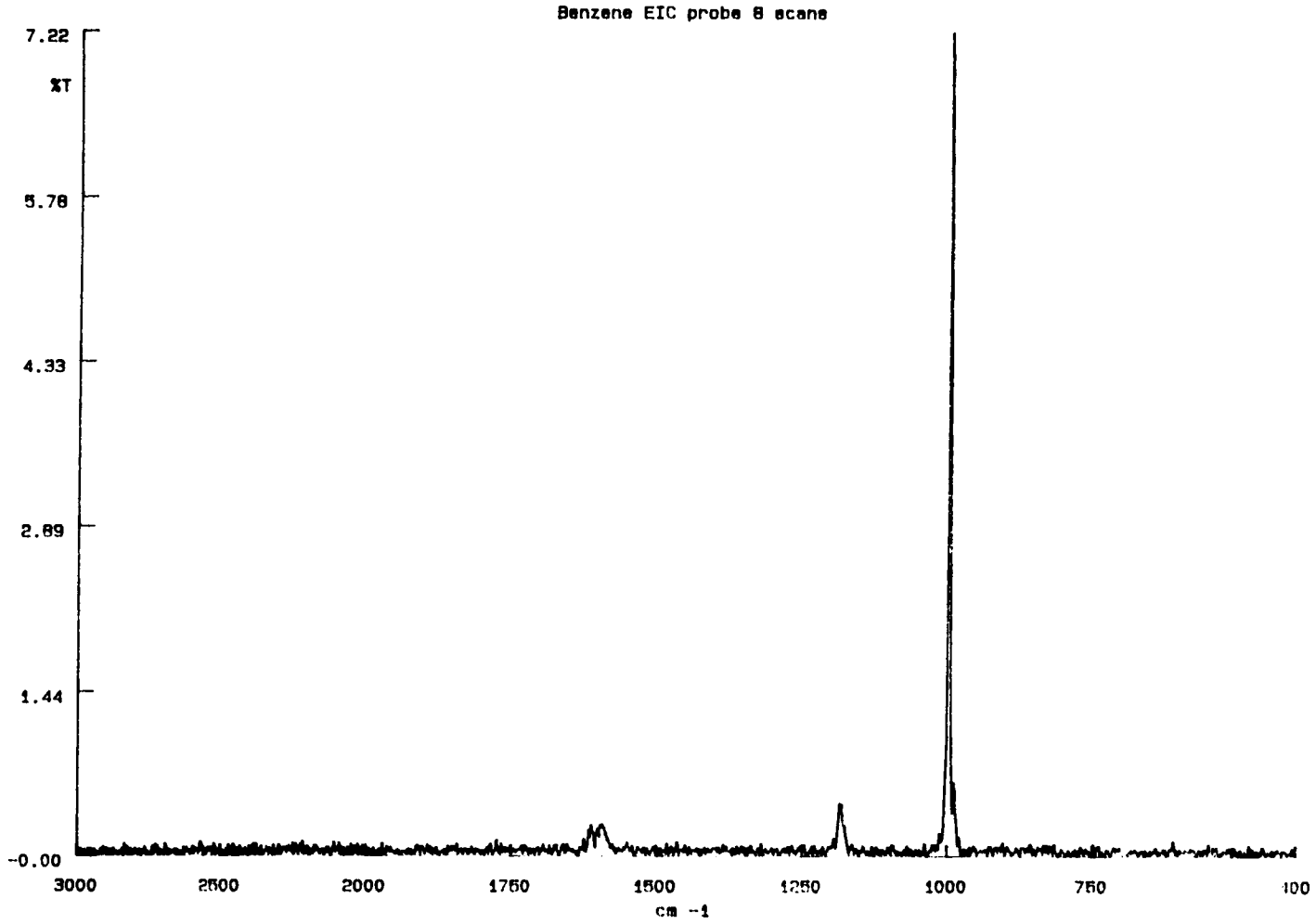


Figure C-16. Benzene - EIC Probe

APPENDIX D

EXPERIMENTAL DATA: CHEMOMETRIC ANALYSIS

This Appendix includes the results of the chemometric measurements for three representative distillation mixtures. The Eastman mixture consists of acetic acid, water, and a proprietary entraining solvent. The UT Chemical Engineering mixture consists of isopropanol and water. The Hercules mixture consists of cumene hydroperoxide, dimethyl benzyl alcohol, acetophenone, and cumene.

All measurements were taken through fibers using a Dow 6-around-1 probe. For the cumene readings, a modified sample cell scheme, illustrated in Figure D-1, was used. Since its fumes are dangerous to breathe, a closed sample cell must be used for cumene.

For each of the three mixtures, twenty samples, each varying in composition, were prepared and Raman spectra were obtained. For the short fiber experiments, two spectra were taken for each sample. One spectrum is the average of 16 interferometer scans (2.2 minutes collection time). The other spectrum is the average of 64 interferometer scans (8.7 minutes collection time). Since the spectrum is the average of many scans, a large number of scans tends to average out the system noise. By quadrupling the number of scans, the noise in the spectrum should be reduced by a factor of two.

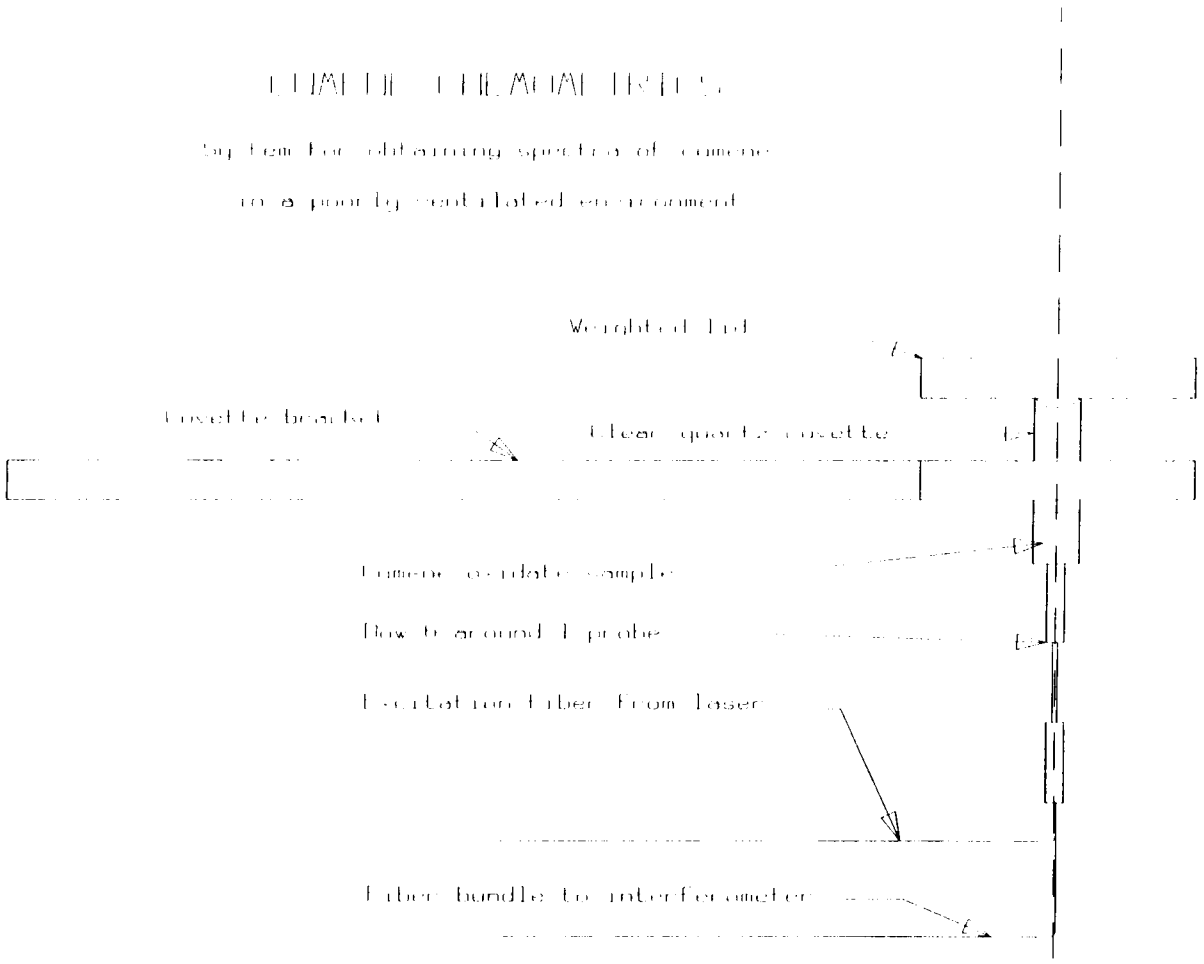


Figure D-1. Cumene Sample Cell

In addition, a long fiber experiment was conducted. Nineteen samples of the Eastman mixture were prepared and Raman spectra were obtained through the 35 meter fiber bundle. Since preliminary measurements had shown that more laser power would be needed to obtain the desired signal to noise performance for 2.2 minute collection times, spectra were only collected for 8.7 minute collection times. The long fiber results predict the performance of the system for 2.2 minute collection times if the laser power were doubled.

In the short fiber experiments, for each mixture and at each collection time, sixteen of the spectra were used as a learning set to create CIRCOM chemometrics program calibration files. For each mixture and collection time, four different bandwidths were used. Hence, a total of 24 different calibration files were created in the learning process.

In the long fiber experiments, for the Eastman mixture at 8.7 minute collection time, sixteen of the spectra were used as a training set to create CIRCOM chemometrics program calibration files. Four different bandwidths were used. Hence, four calibration files were created.

Short fiber experiment bandwidths were as follows. For all mixtures and collection times, calibration files were created using data from the 3750-400 wave number band, and the 1220-620 wave number band (1200-600 for the Eastman mixture). Calibration files were created in the 1200-800 band for the Eastman mixture, the 1220-820 band for the UT mixture, and the 1300-500 band for the Hercules mixture. In addition, calibration files were created for which regions of the spectrum suspected of containing noise but no useful signal were blanked out. For these calibration files, the full 3750-400 band was used for all mixtures. Blanked ranges were 3750-2980, 2700-1820, and 1420-1210 for the Eastman mixture, 3750-2940, 2800-1180, and 780-400 for the UT Chemical Engineering mixture, and 3750-2940, 2800-1300, and 500-400 for the Hercules mixture.

Long fiber experiment bandwidths were as follows. Scheme 1 included the 1000-978, 903-894, and 830-816 wave number bands. Scheme 2 included the entire 1000-816 wave number band. Scheme 3 included the 903-816 wave number band. Scheme 4 included the 903-894 and 830-816 wave number bands.

In the long fiber experiment, the presence of silica Raman features was a major cause for concern. The acetic acid and solvent features both fall within a silica band.

The successive narrowings of bandwidth in the long fiber scheme represent successive exclusions of silica Raman features. Scheme 4 includes only the acetic acid and solvent bands.

After the calibration files were created, CIRCOM was used to analyze each of the four remaining spectra by doing a multiple linear regression analysis of the spectrum against the calibration file. The result is a prediction of the composition that can be compared against the known composition.

Error is defined as the actual percentage less the predicted percentage composition. It is not a percent error in the usual sense. Maximum error is the maximum absolute value of error for all predictions for a set of four samples of a mixture at a given bandwidth and scan rate. RMS error is the square root of the mean of the sum of the squared errors for the same predictions. Standard deviation is the standard deviation of all the squared errors for the same predictions. Note that for the isopropanol mixture, the standard deviation is taken for the actual errors, not the squared errors.

Computed errors in multiple linear regression (MLR) predictions are presented in Tables D-1 for the Eastman mixture through short fibers, D-2 for the Eastman mixture through 35 meter fibers, D-3 for the isopropanol mixture, and D-4 for the Hercules mixture. For the Eastman mixture through short fibers, root mean squared (RMS) error is graphed in Figure D-2, maximum absolute error in Figure D-3, and standard deviation of squared error in Figure D-4. For the Eastman mixture through long fibers, RMS error is graphed in Figure D-5, maximum absolute error in Figure D-6, and standard deviation of squared error in Figure D-7. For the isopropanol mixture through short fibers, RMS error is graphed in Figure D-8, maximum absolute error in Figure D-9, and standard deviation of error in Figure D-10. For the Hercules mixture through short fibers, RMS error is graphed in Figure D-11, maximum absolute error in Figure D-12, and standard deviation of squared error in Figure D-13.

Computed errors in partial least squares (PLS) predictions are presented in Table D-5 for the Eastman mixture through short fibers. RMS error for PLS compared to MLR is graphed in Figure D-14. Maximum error for PLS compared to MLR is graphed in Figure D-15.

Table D-1. Short Fiber Eastman Chemometric Errors

Wave number ranges used for analysis:

3350 wn	3750-400 wn
600 wn	1200-600 wn
400 wn	1200-800 wn
blanked	3750-400 wn with blanking

Errors Observed in Chemometric Predictions:

	16 scans 3350 wn	16 scans 600 wn	16 scans 400 wn	16 scans blanked	64 scans 3350 wn	64 scans 600 wn	64 scans 400 wn	64 scans blanked
Sample AWS 8								
% acid	0.5462	0.1382	0.1962	0.2872	0.4972	0.5692	0.6182	2.1862
% solvent	-0.0065	0.3835	0.3285	0.2395	-1.3625	-0.0195	-0.0895	-1.8625
% water	-0.11793	-0.5222	-0.3358	-0.07935	-0.02179	-0.5466	-0.2116	-0.5575
Sample AWS 11								
% acid	-2.0446	-0.5716	-0.7026	-1.2676	-1.3946	-0.5516	-0.7196	-1.2336
% solvent	2.1821	0.7771	0.9011	1.4421	1.6811	0.9921	1.1131	1.4271
% water	-0.6429	-0.2054	-0.4662	-0.6542	-0.2026	-0.3111	-0.3406	-0.1778
Sample AWS 19								
% acid	-1.6443	-1.3893	-1.3393	-1.4223	-0.2253	-0.4133	-0.5283	-0.8623
% solvent	1.7811	1.5371	1.4891	1.5691	1.0831	0.4951	0.5041	1.1871
% water	-0.6062	-0.1477	-0.4598	-0.6415	-0.4424	-0.1435	-0.3032	-0.2084
Sample AWS 20								
% acid	1.2527	1.6157	2.4207	1.4187	0.4007	0.0277	0.5137	-0.8803
% solvent	-1.0555	-1.4005	-2.1665	-1.2125	0.5725	-0.0275	-0.4395	1.1345
% water	-0.439	-0.2149	-0.257	-0.3817	-0.463	-0.0768	-0.2744	-0.1524
RMS	1.251237	0.928456	1.177010	1.032217	0.867871	0.452069	0.537375	1.170136
Std. Deviation	1.639226	0.984227	1.881869	0.914436	0.915924	0.266689	0.319855	1.406502
Absolute Max.	2.1821	1.6157	2.4207	1.5691	1.6811	0.9921	1.1131	2.1862

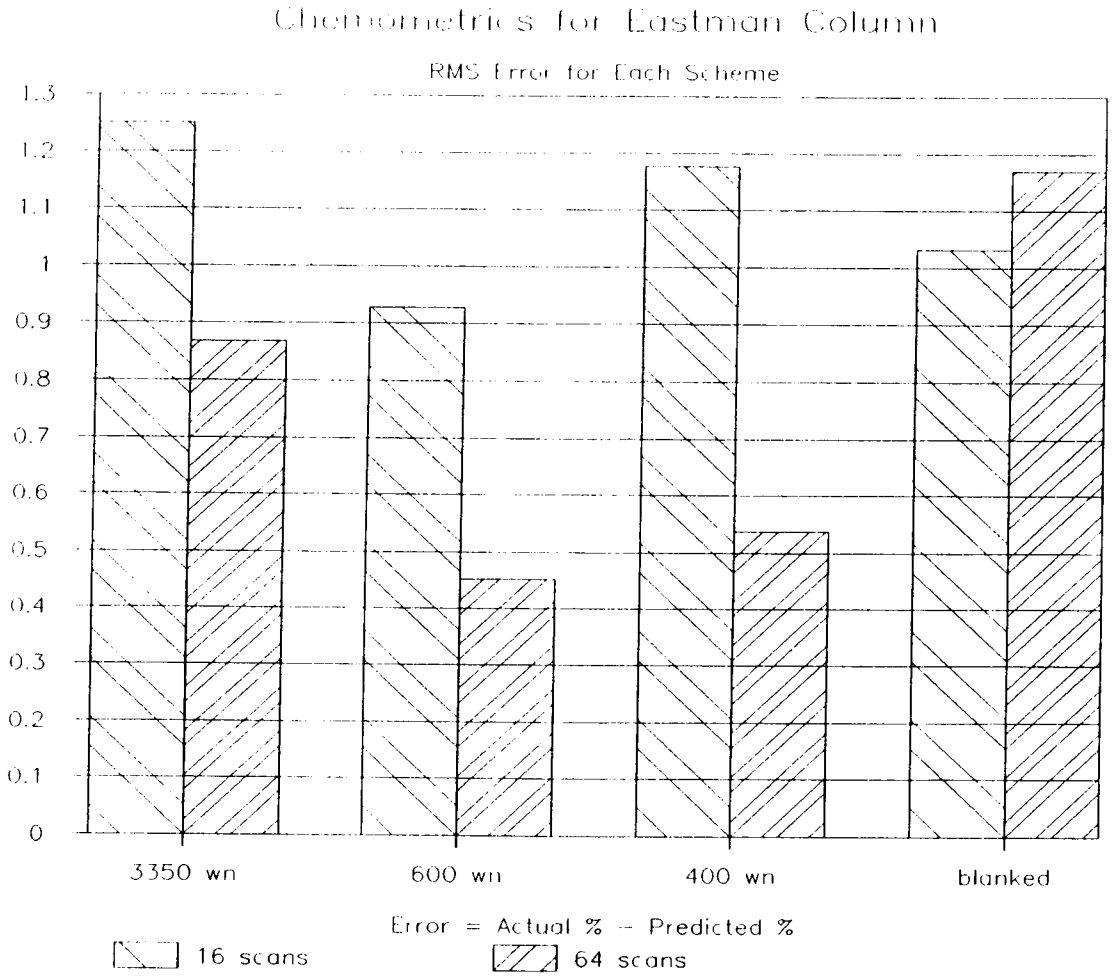


Figure D-2. Short Fiber Eastman RMS Error

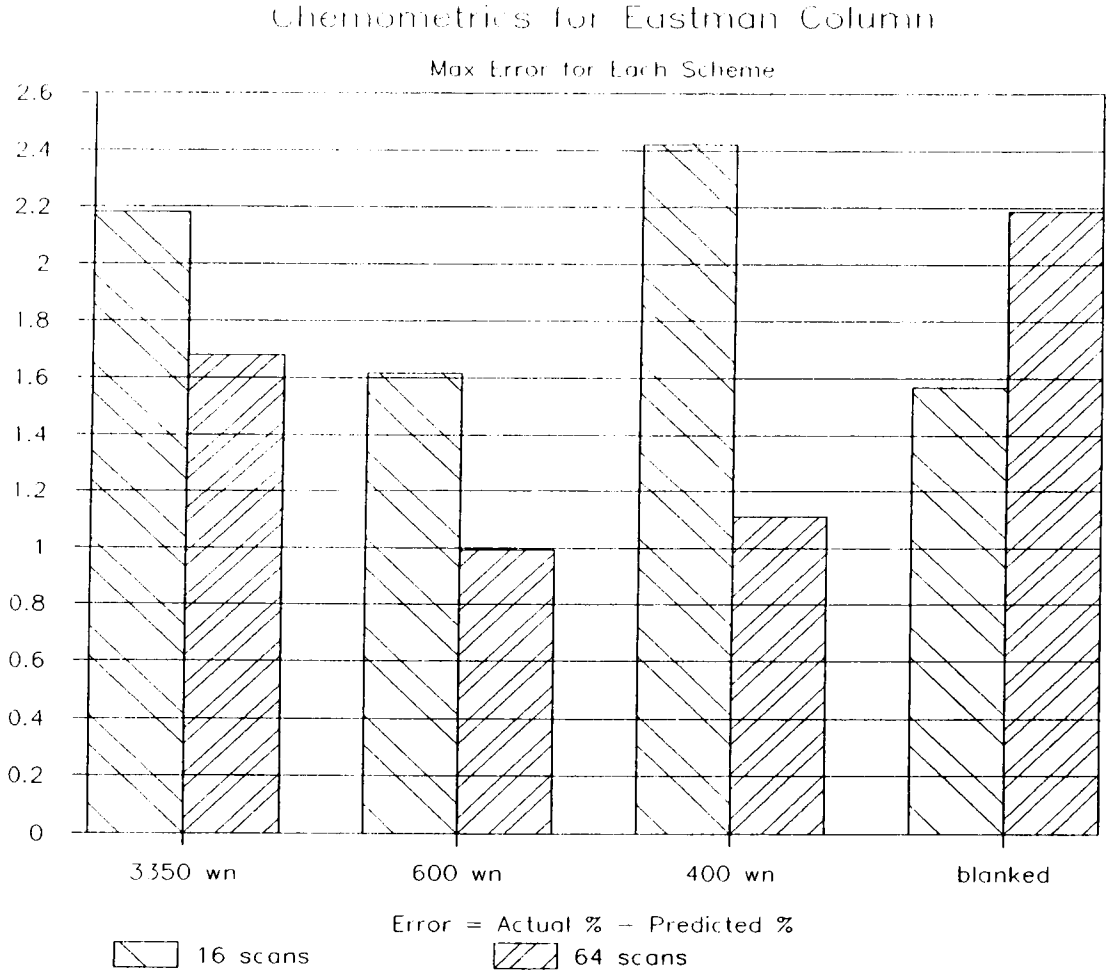


Figure D-3. Short Fiber Eastman Maximum Error

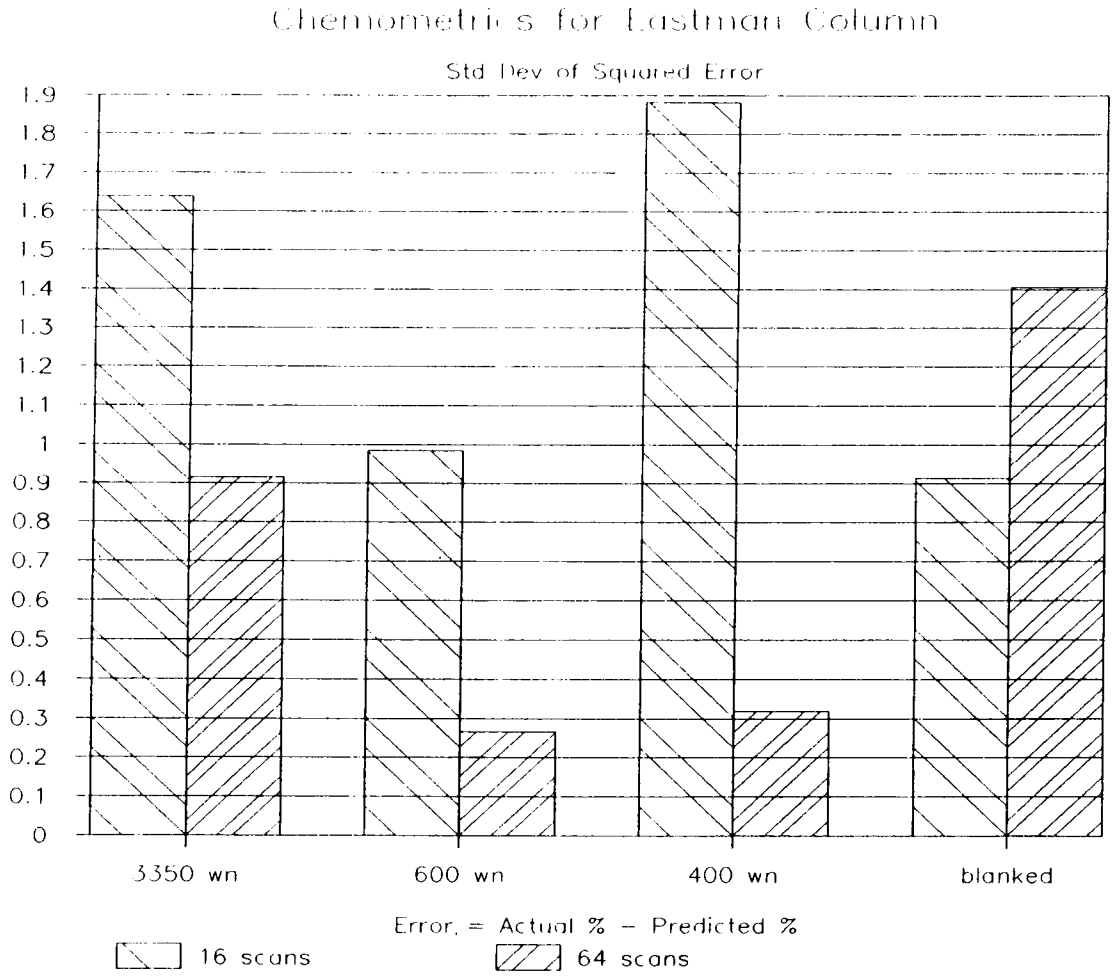


Figure D-4. Short Fiber Eastman Std. Deviation

Table D-2. Long Fiber Eastman Chemometric Errors

Wave number ranges used for analysis:

1 1000-978, 903-894, 830-816 wn
2 1000-816 wn
3 903-816 wn
4 903-894, 830-816 wn

Errors Observed in Chemometric Predictions:

	64 scans 1	64 scans 2	64 scans 3	64 scans 4
Sample SAW3				
% acid	-1.481	0.219	-0.639	-1.366
% solvent	-3.214	-6.569	-5.342	0.100
% water	4.793	6.364	5.877	4.559
Sample SAW11				
% acid	-0.189	-2.953	-2.076	-0.043
% solvent	1.363	4.148	3.935	2.045
% water	-1.130	-1.507	-1.989	-0.274
Sample SAW15				
% acid	-0.022	-0.211	-0.821	-0.227
% solvent	-2.436	-0.807	-0.510	1.293
% water	2.389	1.685	1.418	2.282
RMS	2.364	3.581	3.165	1.939
Std. Deviation	6.919	16.356	12.393	6.284
Absolute Max.	4.793	6.569	5.877	4.559

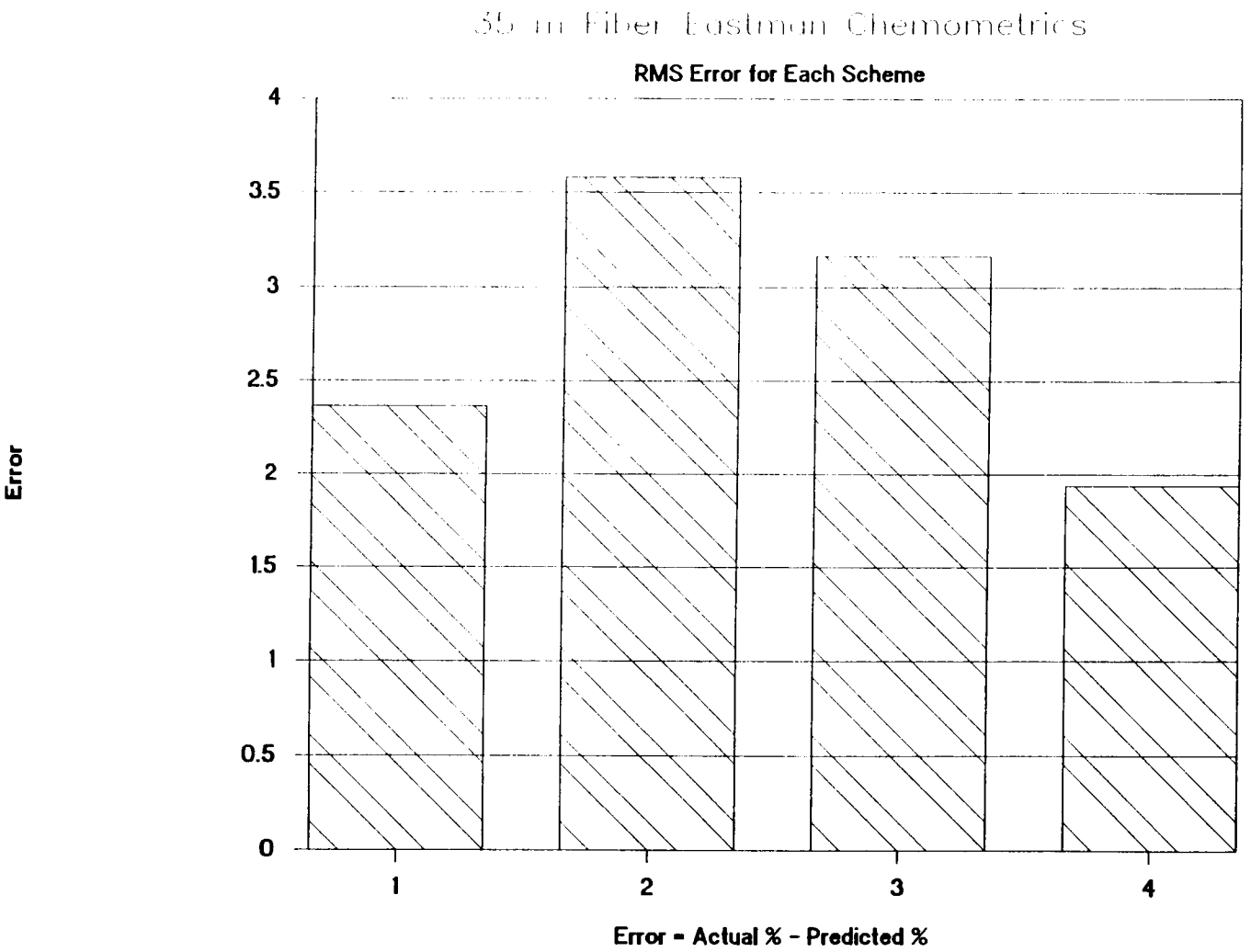


Figure D-5. Long Fiber Eastman RMS Error

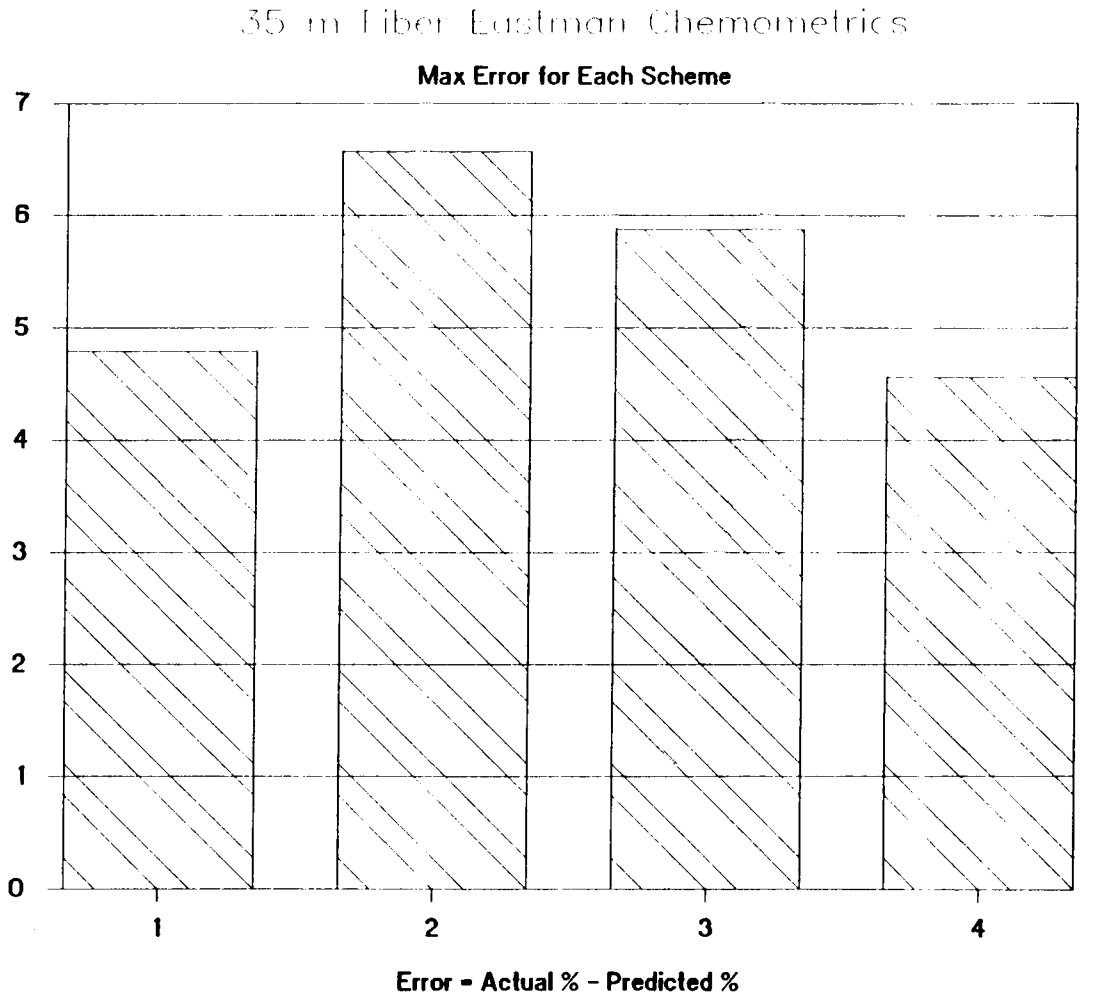


Figure D-6. Long Fiber Eastman Maximum Error

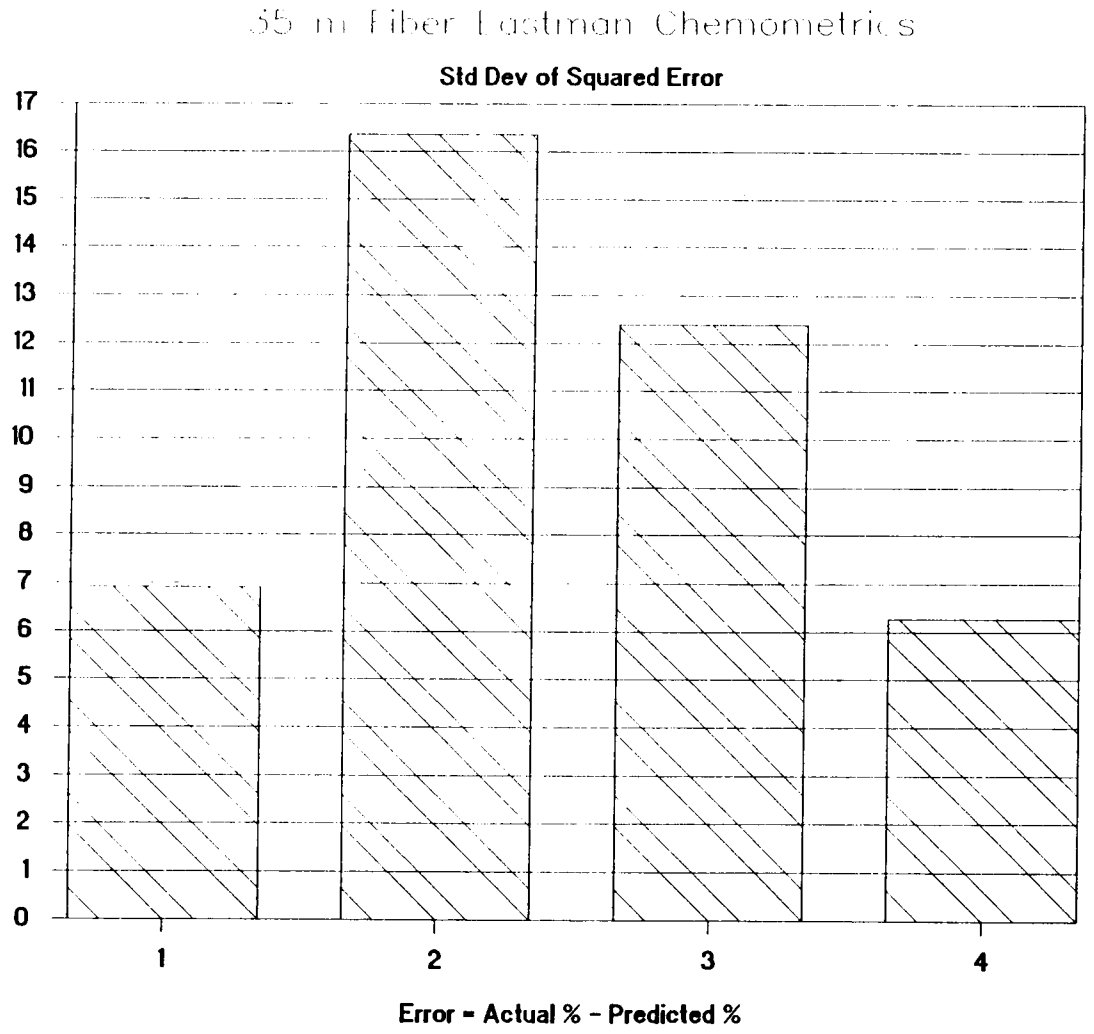


Figure D-7. Long Fiber Eastman Std. Deviation

Table D-3. Short Fiber Isopropanol Chemometric Errors

Wave number ranges used for analysis:
- 3750-400 wn
- 1220-620 wn
- 1220-820 wn
- 3750-400 wn with blanking

Errors Observed in Chemometric Predictions:

	16 scans 3350 wn	16 scans 600 wn	16 scans 400 wn blanked	16 scans	64 scans 3350 wn	64 scans 600 wn	64 scans 400 wn blanked	64 scans
Sample IP2								
% alcohol	-0.8867	-1.1547	-0.7727	-0.7677	-1.4027	-1.2297	-1.0047	-0.6057
% water	0.8877	1.1557	0.7727	0.7687	1.4037	1.2297	1.0047	1.0557
Sample IP6								
% alcohol	0.7818	0.6708	1.0208	0.6278	0.4038	0.6248	0.5298	0.5108
% water	-0.7818	-0.6708	-1.0208	-0.6278	-0.4038	-0.6248	-0.5298	-0.5098
Sample IP15								
% alcohol	2.7046	2.0576	1.7066	2.0736	0.8276	0.6186	0.4036	0.6606
% water	-2.7046	-2.0576	-1.7066	-2.0736	-0.8276	-0.6186	-0.4036	-0.6606
Sample IP18								
% alcohol	0.4496	0.3186	0.2736	0.1866	0.1396	-0.0094	-0.2904	-0.1544
% water	-0.4496	-0.3186	-0.2736	-0.1866	-0.1396	0.0094	0.2904	0.1544
RMS	1.493002	1.237016	1.075457	1.153220	0.842294	0.755859	0.619946	0.677430
Std. Deviation	0.880227	0.652910	0.516293	0.702994	0.477780	0.431448	0.271929	0.323173
Absolute Max.	2.7046	2.0576	1.7066	2.0736	1.4037	1.2297	1.0047	1.0557

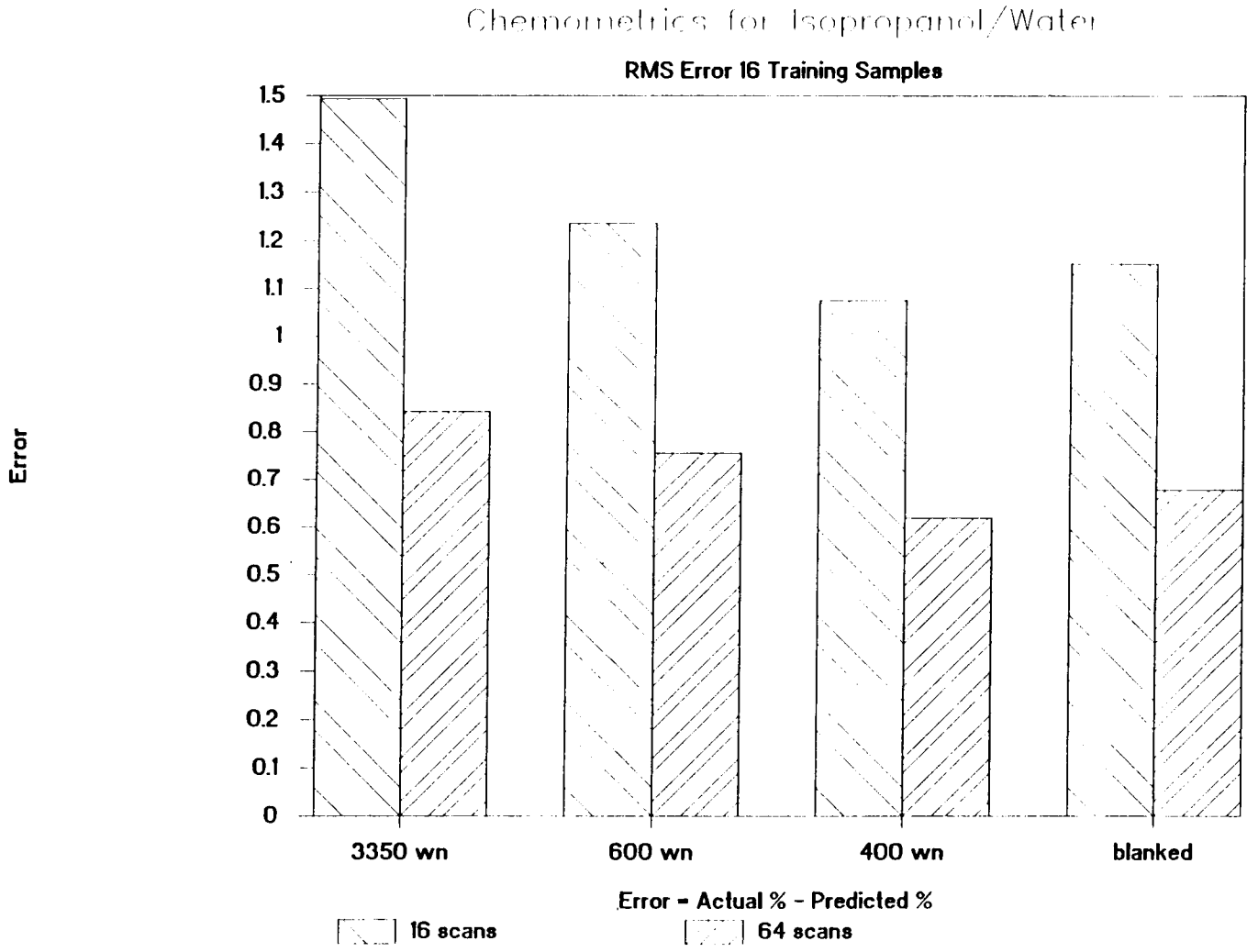


Figure D-8. Short Fiber Isopropanol RMS Error

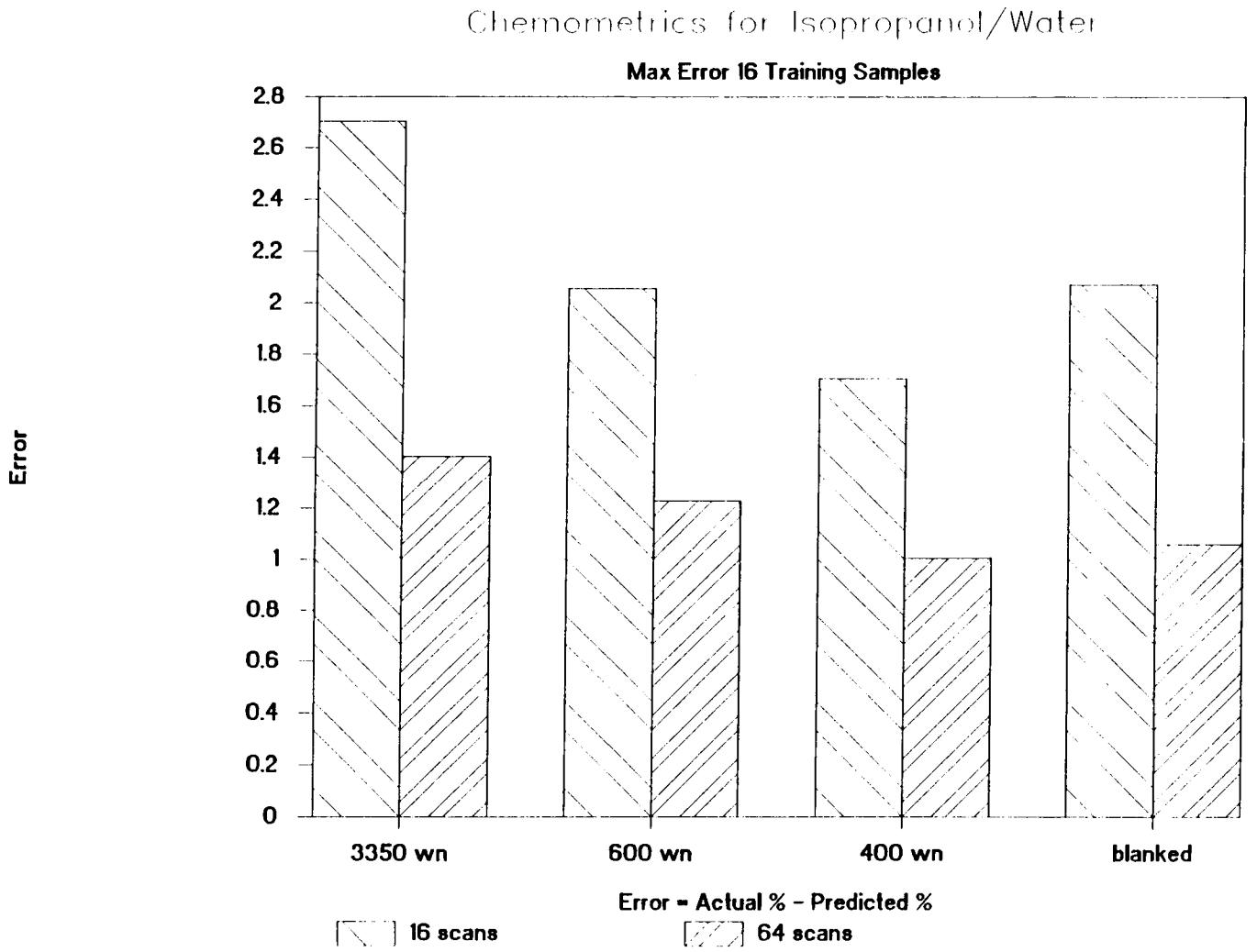


Figure D-9. Short Fiber Isopropanol Maximum Error

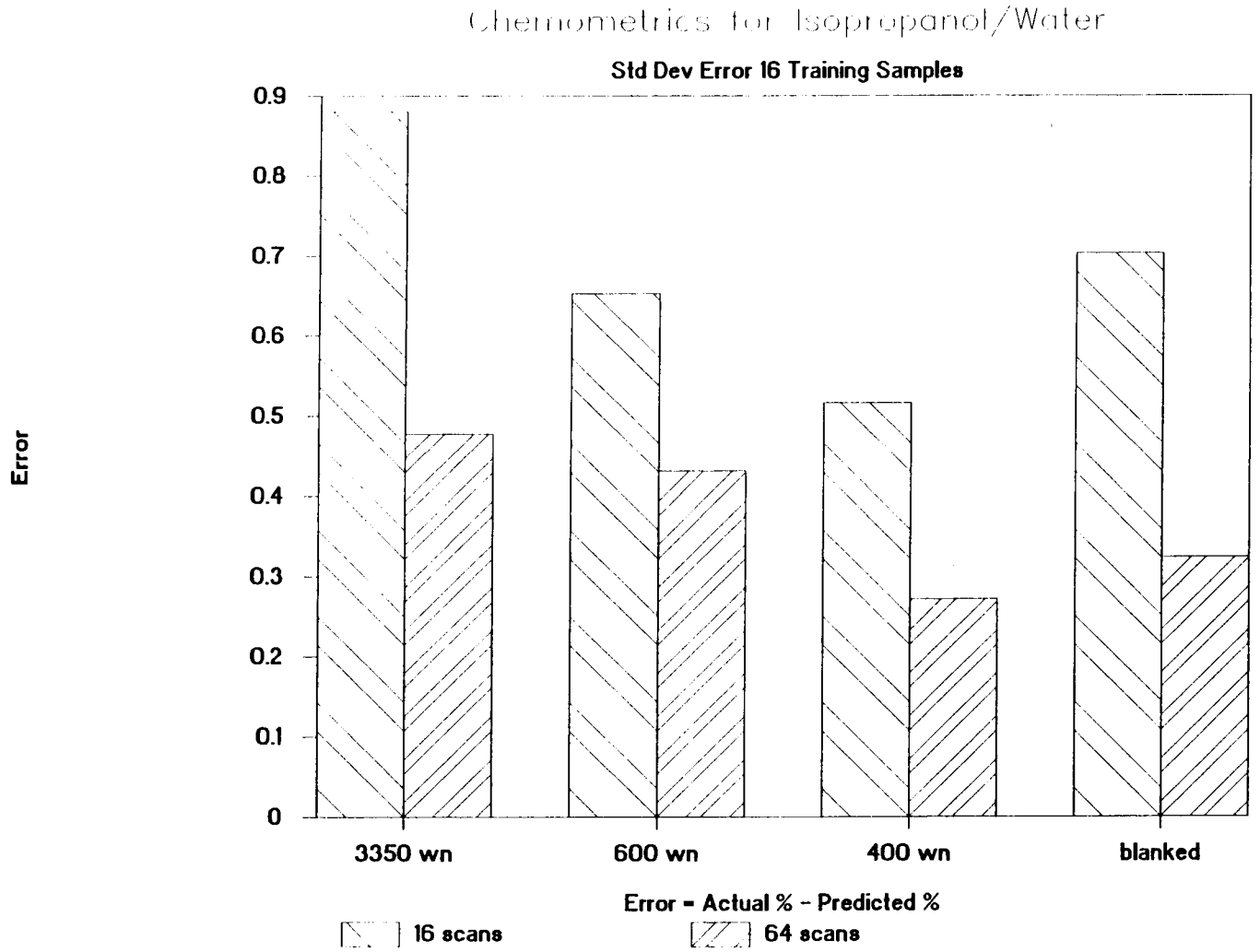


Figure D-10. Short Fiber Isopropanol Std. Deviation

Table D-4. Short Fiber Hercules Chemometric Errors

The four components are:	Wave number ranges used for analysis:
Cumene hydroperoxide (CHP)	3750-400 wn
Dimethyl benzyl alcohol (DMBA)	1300-500 wn
Acetophenone (ACP)	1220-620 wn
Cumene (CM)	3750-400 wn with blanking

Errors Observed in Chemometric Predictions:

	16 scans 3350 wn	16 scans 800 wn	16 scans 600 wn	16 scans blanked	64 scans 3350 wn	64 scans 800 wn	64 scans 600 wn	64 scans blanked
Sample C1								
% CHP	1.96	1.821	1.414	2.0886	0.488	0.402	0.312	0.217
% DMBA	0.265	0.5389	0.6585	0.5538	0.9007	0.9718	0.958	0.9237
% ACP	-1.5318	-1.4848	-1.3796	-1.4729	-0.2646	-0.3286	-0.4346	-0.3296
% CM	-0.8732	-0.8782	-0.6832	-0.8392	-0.4352	-0.5202	-0.0272	-0.2602
Sample C15								
% CHP	-0.2637	-0.3387	-0.4947	-0.3467	-0.3597	-0.9507	-1.0127	-1.0617
% DMBA	-0.917	-0.8304	-0.7555	-0.8304	-1.1246	-1.0929	-1.126	-1.0801
% ACP	3.06	3.2236	3.1557	3.1903	1.5821	1.803	1.9124	1.7427
% CM	-1.9292	-2.0502	-1.8032	-1.9962	-1.1792	-0.8962	-1.4442	-0.8462
Sample C19								
% CHP	-5.231	-4.986	-4.699	-5.044	-5.657	-5.446	-5.507	-5.515
% DMBA	2.0219	2.0848	2.0007	2.1045	2.1847	2.0637	1.9836	2.0177
% ACP	1.8226	1.7126	1.6459	1.705	1.412	1.3204	1.2992	1.2187
% CM	1.2801	1.1801	1.0691	1.1911	1.4511	1.4321	1.2301	1.5651
Sample C20								
% CHP	-0.0506	-0.7426	-1.1296	-0.8146	-0.9966	-0.9966	-1.1036	-1.0736
% DMBA	0.4695	1.0545	1.1701	1.0632	1.1472	1.0379	1.0049	1.0494
% ACP	0.2397	0.2324	0.1986	0.2178	0.3325	0.2409	0.2632	0.2239
% CM	-0.9554	-0.5574	-0.1904	-0.5284	-0.9264	-0.7254	-0.8554	-0.6764
RMS	1.914121	1.896408	1.793821	1.917557	1.780950	1.732116	1.763034	1.737565
Std. Deviation	6.559248	6.045857	5.409558	6.159987	7.534793	6.972712	7.115486	7.158433
Absolute Max.	5.231	4.986	4.699	5.044	5.657	5.446	5.507	5.515

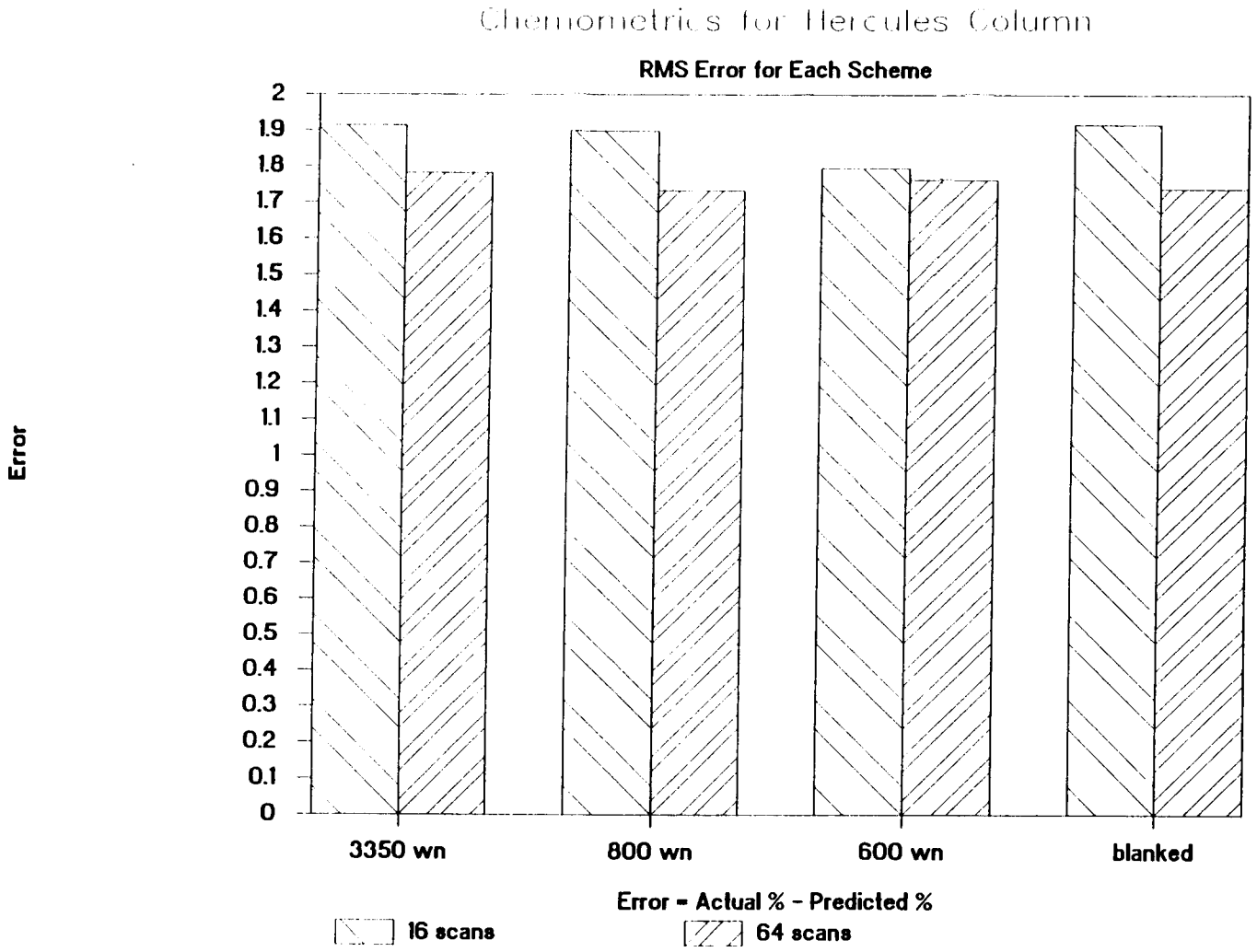


Figure D-11. Short Fiber Hercules RMS Error

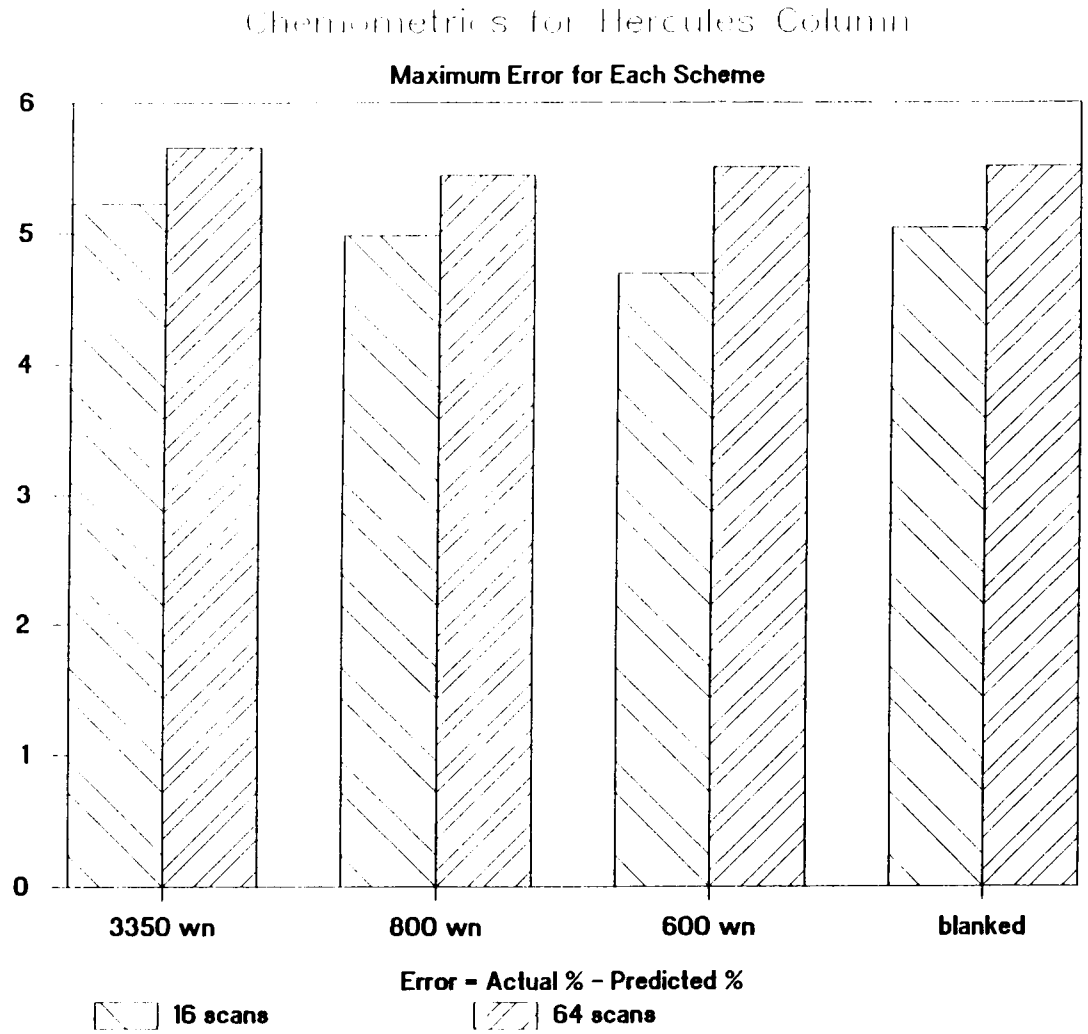


Figure D-12. Short Fiber Hercules Maximum Error

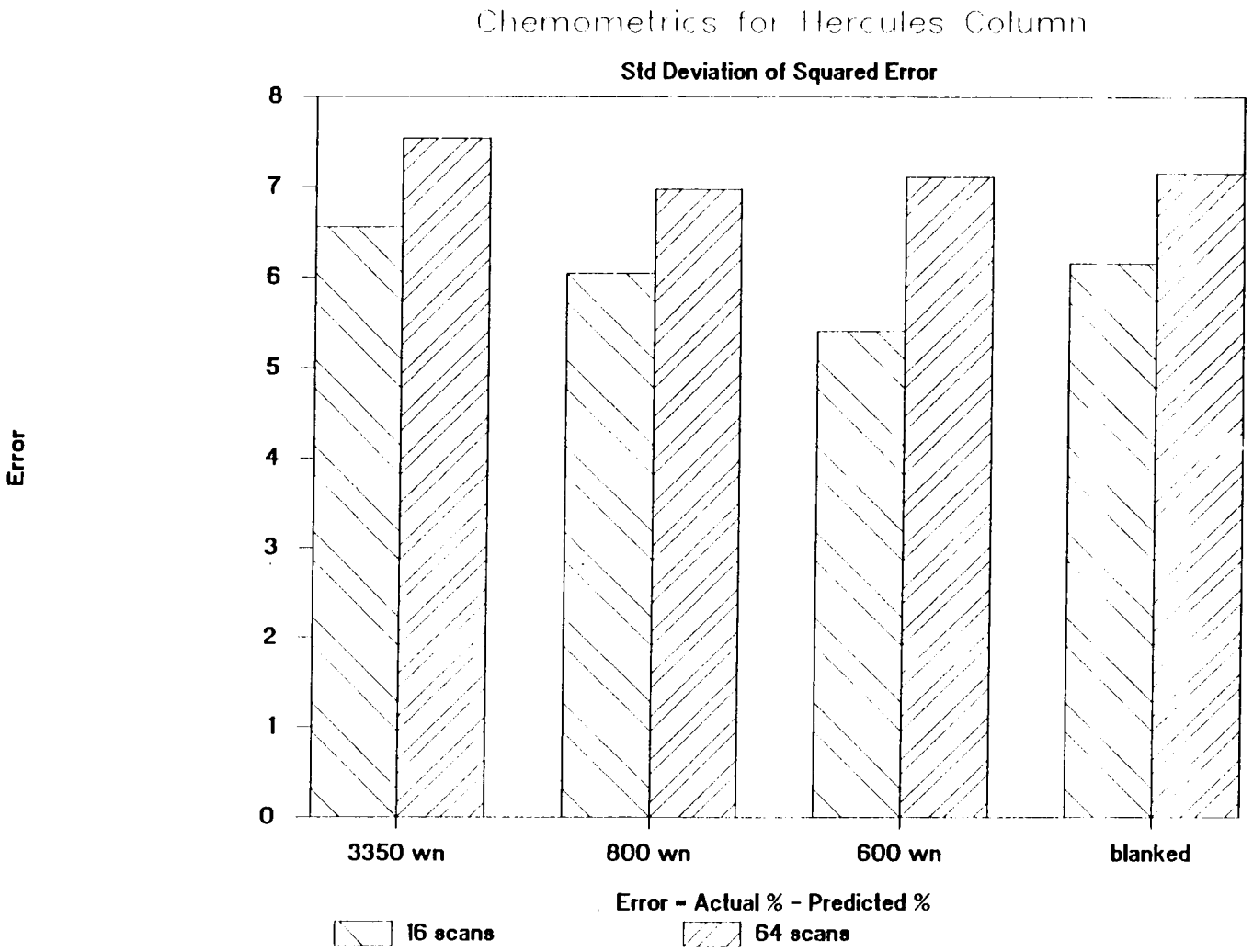


Figure D-13. Short Fiber Hercules Std. Deviation

Table D-5. Eastman Partial Least Squares Errors

For the Eastman mixture through short fiber scanning ranges:

1 region: 993-561 wave numbers
3 regions: 2978-2794, 1820-1634, 997-544 wave numbers
4 regions: 1052-1004, 992-863, 841-812, 664-568 wave numbers

Errors Observed in Partial Least Squares Predictions:

	16 scans 1 region	16 scans 4 regions	16 scans 3 regions
Sample AWS8			
% acid	-2.153	-0.5767	-1.3537
% acid	3.4732	1.5475	1.658
% acid	-0.5031	-0.6122	-0.3463
Sample AWS11			
% acid	-1.8791	-2.0467	-4.7134
% acid	4.6288	3.8096	6.3912
% acid	-0.4134	-0.2195	-0.1057
Sample AWS19			
% acid	1.077	0.4021	-0.4403
% acid	0.1269	0.9926	1.1871
% acid	-0.2246	-0.2491	-0.1382
Sample AWS19			
% acid	0.5283	1.6513	-0.3298
% acid	0.6748	-1.0999	1.2318
% acid	-0.1278	-0.1536	-0.1019
RMS	1.916085	1.5005969	2.433015
Absolute Max.	4.6288	3.8096	6.3912

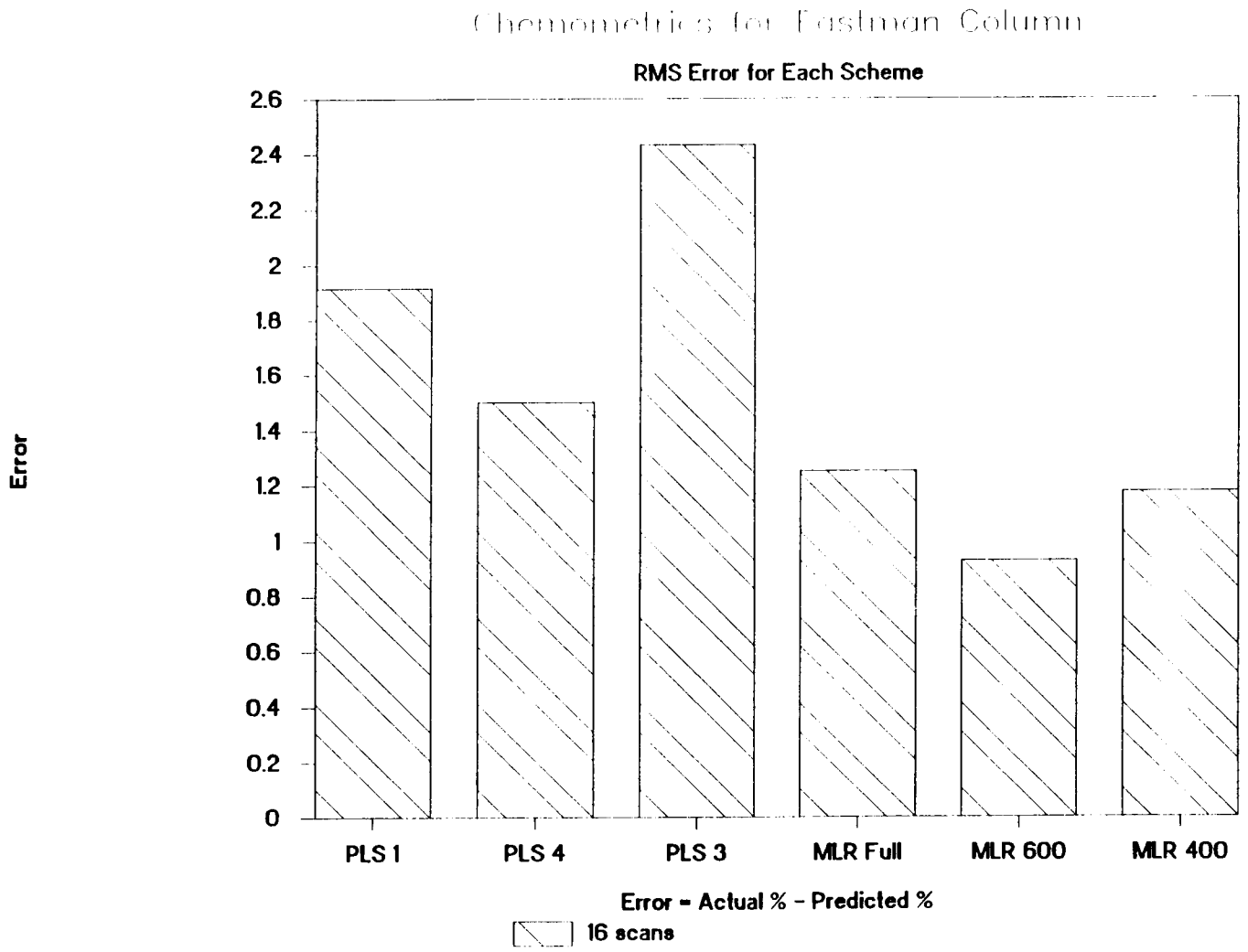


Figure D-14. RMS Error PLS vs. MLR

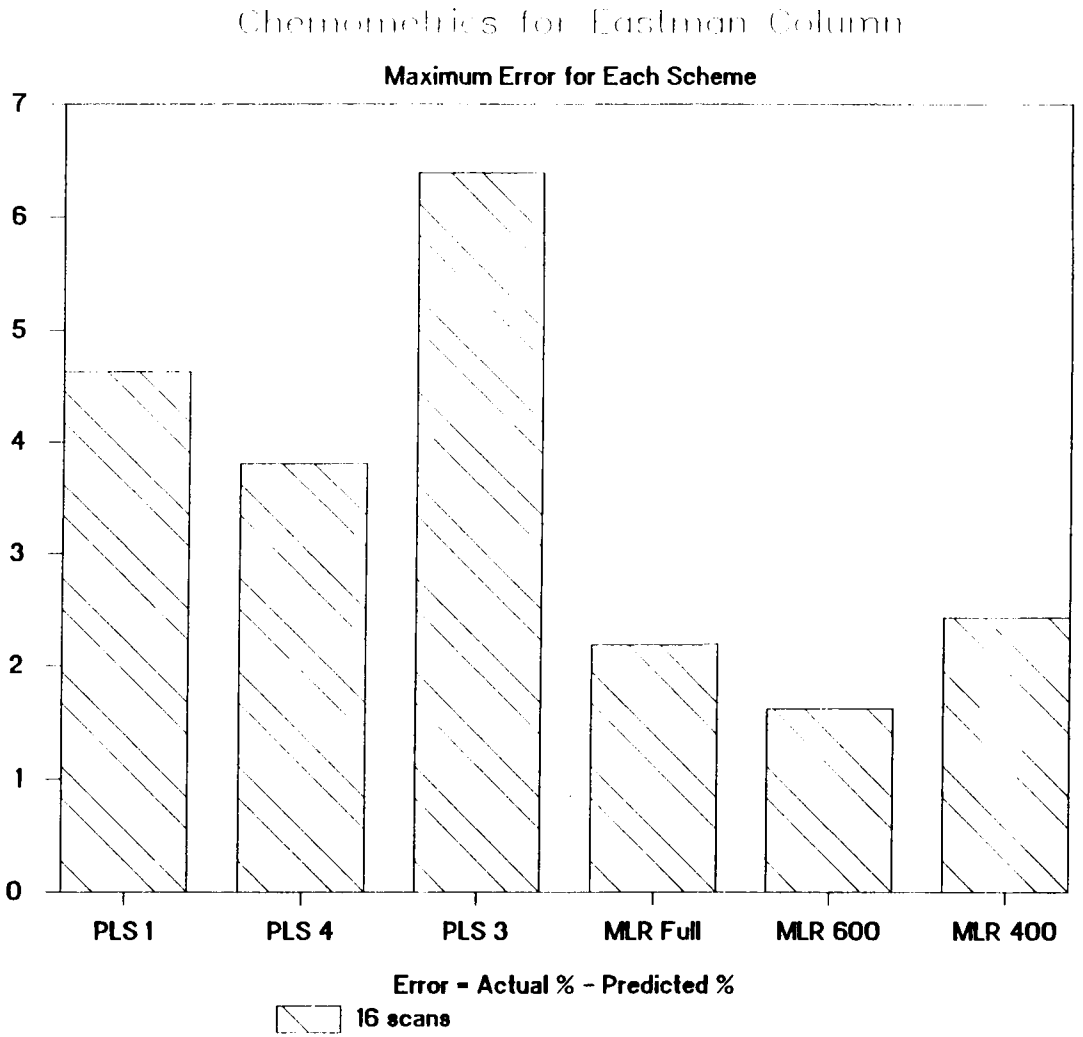


Figure D-15. Maximum Error PLS vs. MLR

APPENDIX E

EXPERIMENTAL DATA: CENTER OF SCATTERING

Procedure

The equipment for this experiment was set up as shown in Figure E-1. Its purpose was to observe the Raman scattering spectrum for benzene when excited by infrared radiation at 1.064 microns. Detailed observations of the spectra were made with the CDS software provided with the Perkin-Elmer computer.

The sample cell consisted of a 50 ml graduated cylinder filled with benzene. The fiber optic probe was immersed in the benzene. Great care was used to align the center of the probe with the center of the cylinder. Note the dimension X on the diagram. This was the distance from the bottom face of the fiber optic probe and the bottom of the graduated cylinder.

The questions addressed by this experiment include the following: Does the X dimension affect the observed spectrum? How do signal intensity and signal to noise ratio vary as X varies? Is there an optimum value for X?

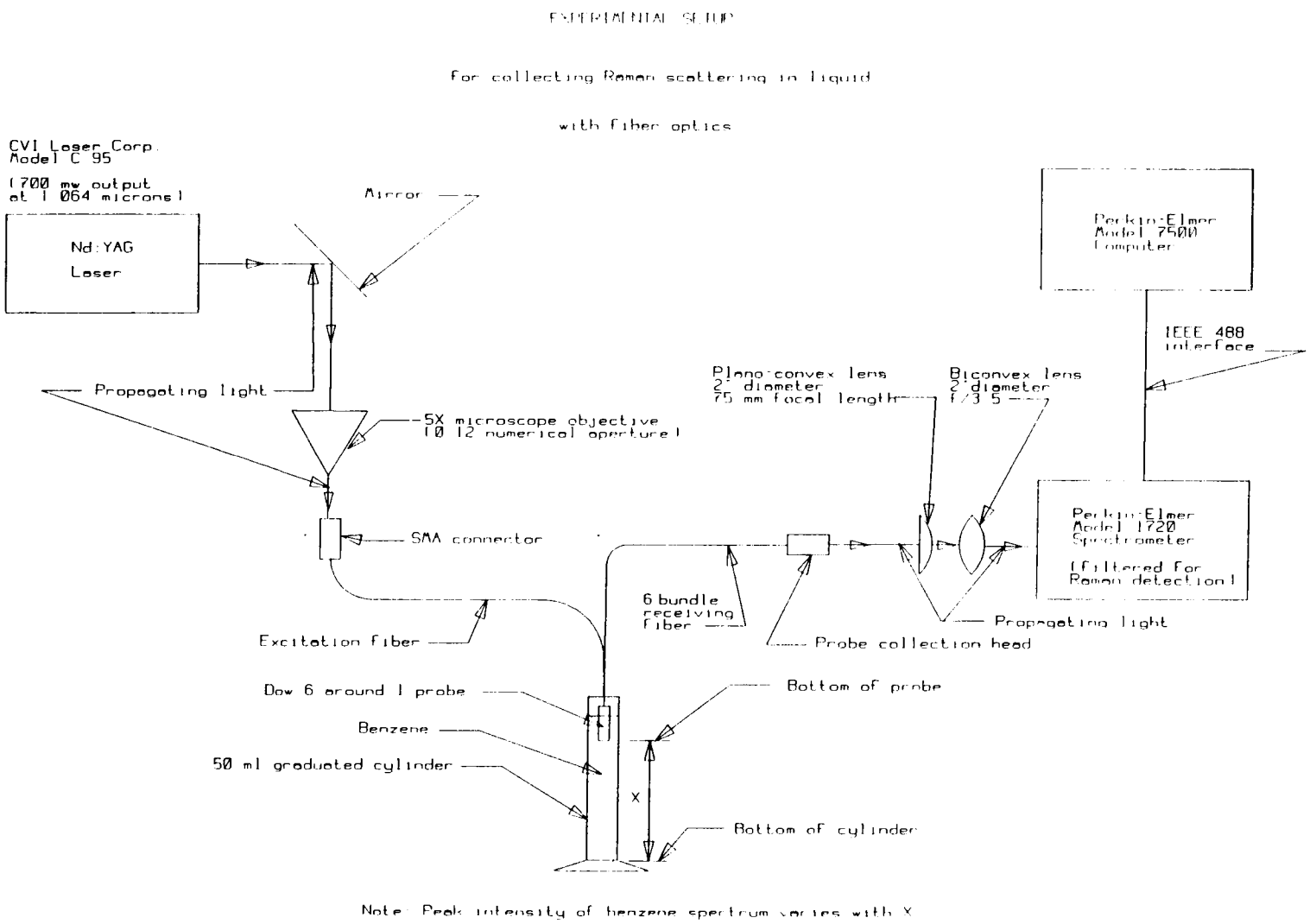


Figure E-1. Benzene Profile Experiment

The experimental procedure was as follows: The end of the probe was aligned with a graduation mark on the cylinder. At each alignment a spectrum was taken. The following conditions were held constant: Radiation power at the sample was 725 milliwatts. Resolution was two wave numbers. The final spectrum was the average of 8 scans. The total time to generate each spectrum was 1.1 minutes.

Observed Data

Plots of two representative spectra are shown in Figure E-2. Both plots show the strong benzene line at 980 wave numbers and the weaker benzene bands in the 1200 and 1600 wave number area. For spectra taken with the probe close to the bottom of the vessel, some additional features appear. These consist of a band with a peak at 2954 wave numbers and glass features in the 1250-450 wave number band. These are spectral features of the bottom of the vessel. Thus the spectrum taken with the probe at the 2 ml mark shows the benzene spectrum with the spectrum of the bottom of the vessel superimposed.

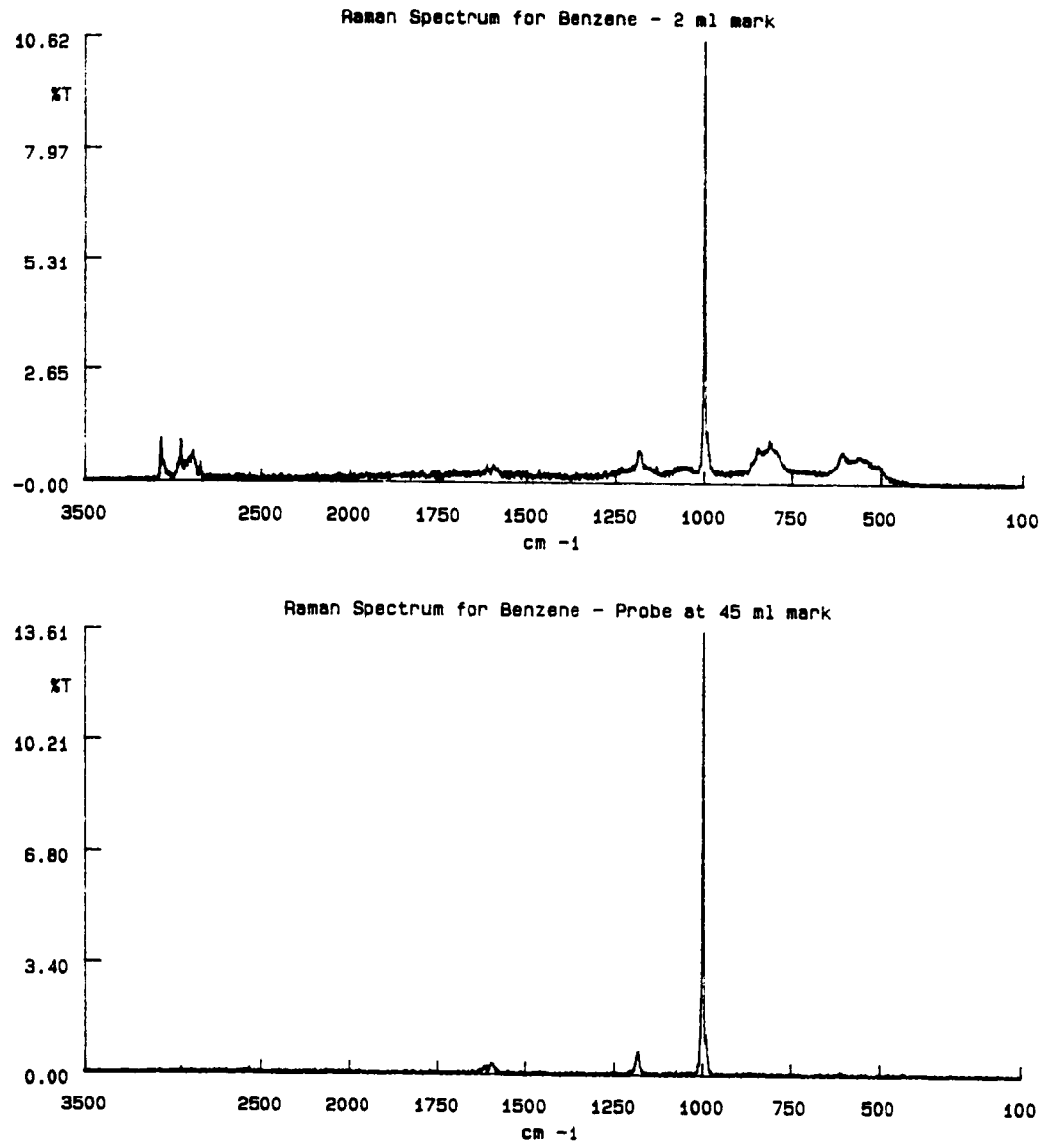


Figure E-2. Representative Benzene Profile Spectra

Observed data are recorded in Table E-1. Each line in the table summarizes a different observation. The meaning of the data in each column is as follows: The ml mark is the mark with which the probe was aligned for a given observation. The distance is the corresponding distance of the bottom of the probe to the bottom of the vessel. The desired signal intensity is the intensity of the benzene line at 980 wave numbers. The two undesired signals are the intensities of the two bands with peaks at 2954 and 814 wave numbers. These are artifacts of the bottom of the vessel. As such they are not random. Hence, strictly speaking, they are not noise. They are undesired signals. The ratio of the desired signal to the undesired signal is calculated for both undesired signals. For this experiment, noise is defined as the peak intensity in the 1500-1250 wave number band. The signal to noise ratio is the ratio of the 980 wave number benzene peak to the peak intensity (at whatever wave number it occurs) in the noise band as defined above.

In order to clarify the tabulated data, several graphs are included (Figures E-3, E-4, E-5 and E-6). Figure E-3 shows the noise and undesired signals plotted against the X dimension. Note that the intensities tend to decrease with increasing X out to a distance of about 40 mm. Then they tend to level off. The most important measure in an infrared

Table E-1. Benzene Intensity Profile

6-around-1 Probe Immersed in Benzene
Probe Tip at Distance X from Bottom of Sample Cell

	A	B	C	D	E	F	G
		Undesired Signals:				Noise:	
Mark	X	Desired Signal	2954 cm-1		814 cm-1	Peak	S/N ratio
ml	mm	Intensity	Intensity	A/B	Intensity	A/D	Intensity	A/F
1	2.7	3.51	0.81	4.333333	1.8	1.95	0.5	7.02
2	5.4	10.62	1.02	10.41176	1.07	9.925233	0.35	30.34285
3	8.1	12.9	1.07	12.05607	0.6	21.5	0.24	53.75
4	10.8	13.9	1.03	13.49514	0.34	40.88235	0.2	69.5
5	13.5	14.3	1	14.3	0.23	62.17391	0.19	75.26315
6	16.2	14.47	0.85	17.02352	0.21	68.90476	0.19	76.15789
7	18.9	14.49	0.78	18.57692	0.22	65.86363	0.16	90.5625
8	21.6	14.5	0.74	19.59459	0.09	161.1111	0.14	103.5714
9	24.3	14.91	0.63	23.66666	0.1	149.1	0.15	99.4
10	27	14.81	0.55	26.92727	0.04	370.25	0.16	92.5625
15	40.5	14.9	0.33	45.15151	0.09	165.5555	0.14	106.4285
20	54	14.9	0.33	45.15151	0.09	165.5555	0.15	99.33333
25	67.5	14.7	0.22	66.81818	0.06	245	0.15	98
30	81	14.24	0.19	74.94736	0.04	356	0.14	101.7142
35	94.5	13.94	0.14	99.57142	0.03	464.6666	0.15	92.93333
40	108	13.72	0.06	228.6666	0.05	274.4	0.15	91.46666
45	121.5	13.61	0.13	104.6923	0.01	1361	0.12	113.4166

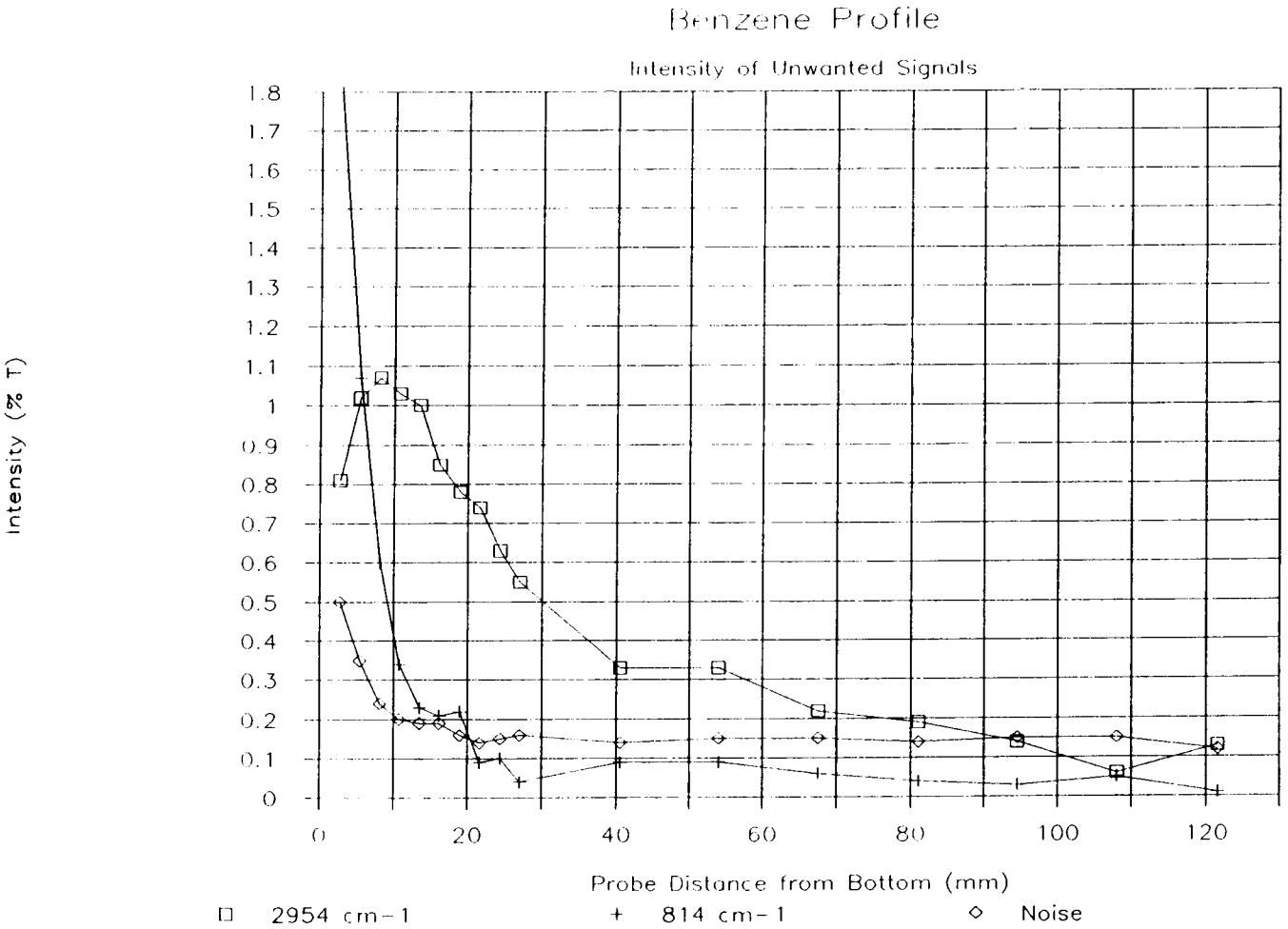


Figure E-3. Undesired Signals in Benzene Profile

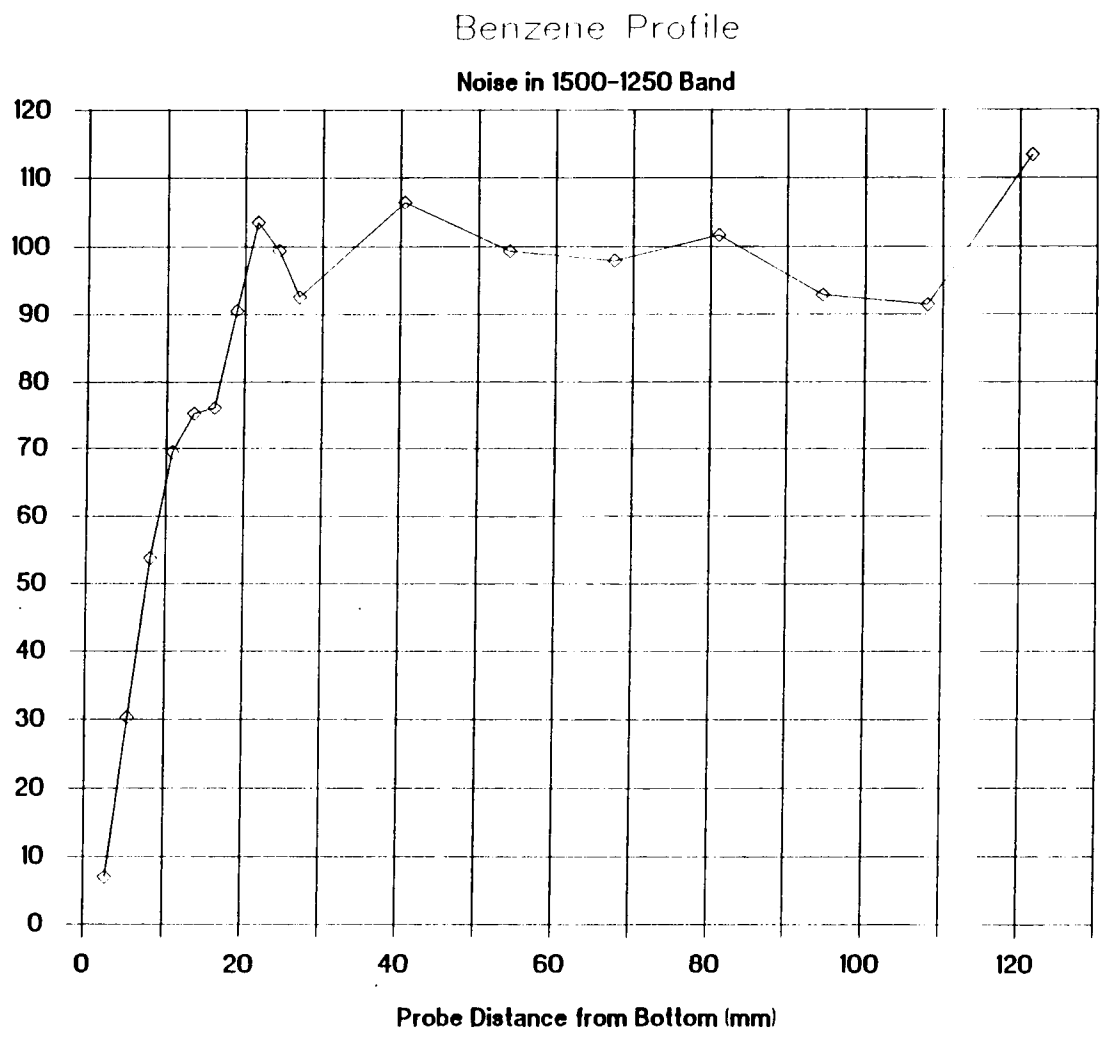


Figure E-4. Noise in Benzene Profile

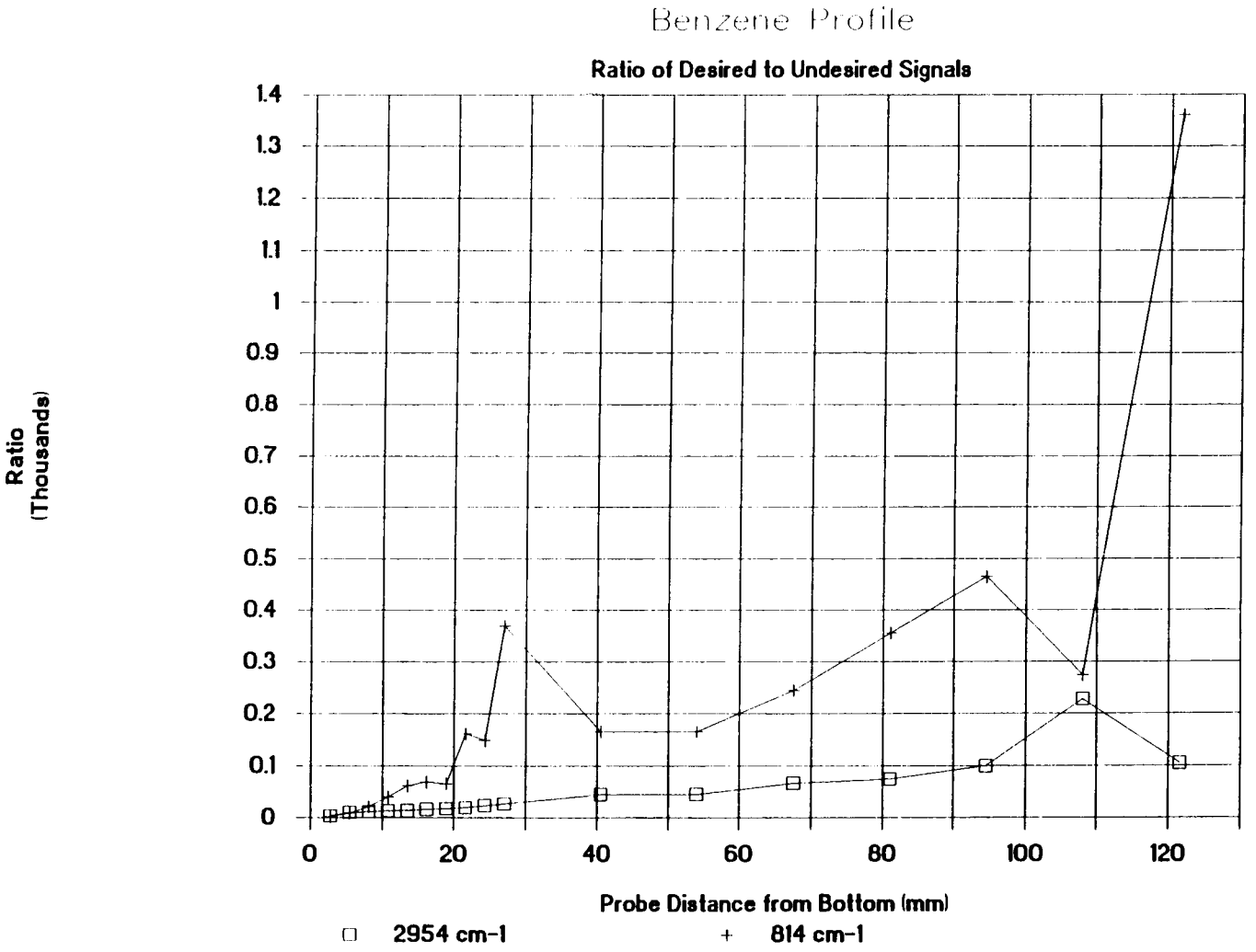


Figure E-5. Ratio of Desired to Undesired Signals

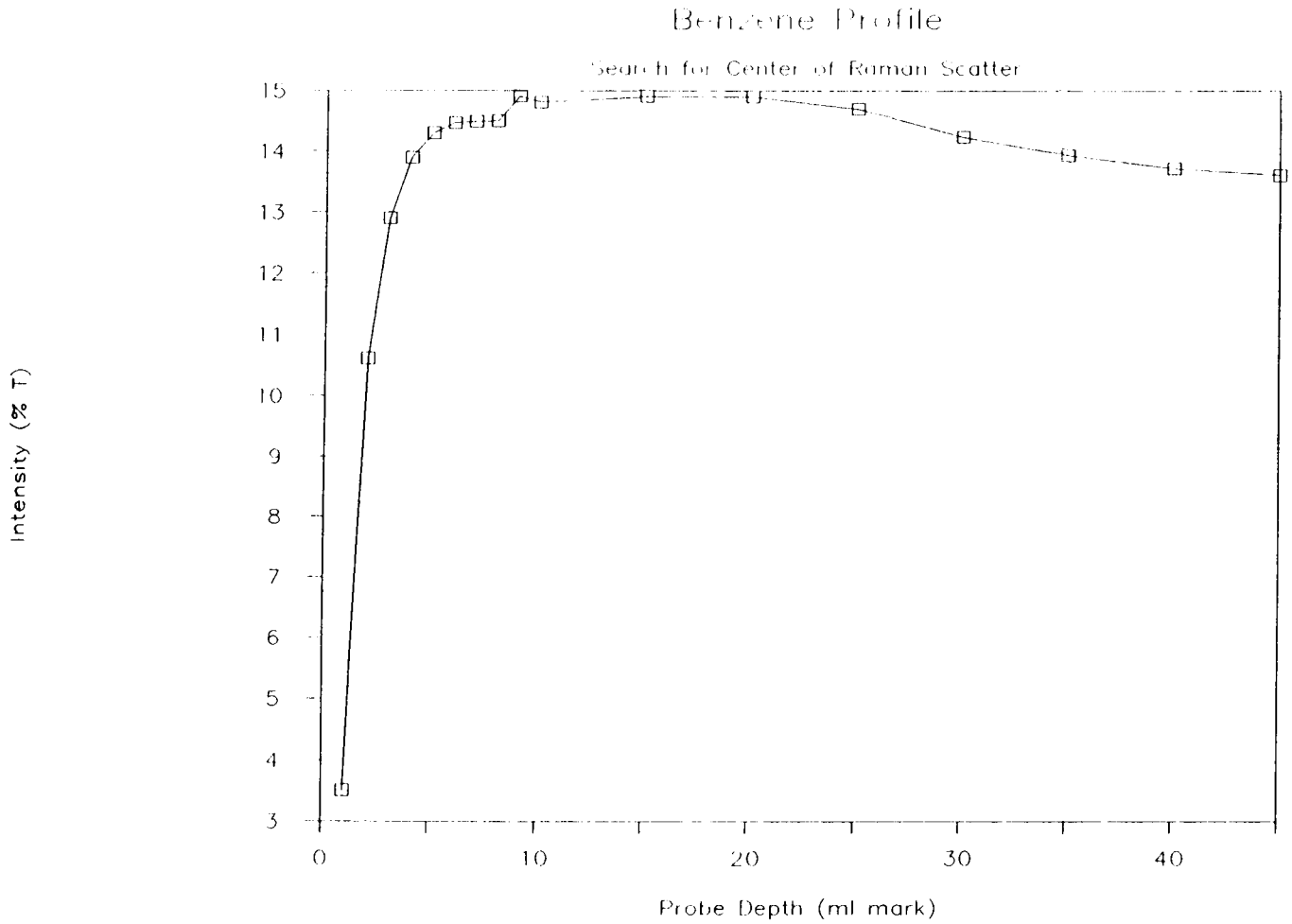


Figure E-6. Benzene Intensity Versus Probe Depth

Raman system is the signal to noise ratio. This is indicated in Figure E-4. The signal to noise ratio increases rapidly from low values of X out to 40 mm. At 40 mm it reaches a local maximum, and is not very sensitive to distance for higher values of X. Figure E-5 shows ratios of desired to undesired signals. Finally, Figure E-6 show the intensity of the desired signal as a function of X. The maximum value of the benzene peak occurs when X is 40 mm.

Conclusions

In taking Raman spectra of benzene, probe placement matters. The probe end should be at least 40 mm from the bottom of the vessel. This keeps the observed spectrum relatively free of undesired signals.

While the X dimension should be about 40 mm for best results (for benzene), the exact dimension is not critical. A variation of 10 mm either way has almost no effect on the signal intensity and minimal effect on the signal to noise ratio.

When the probe is very close to the bottom of the vessel, the intensity of the benzene drops off (indicating that less scattering is occurring in the benzene) and the vessel artifacts become prominent (indicating that a large

proportion of unscattered light is passing through the thin layer of benzene and interacting with the vessel). These effects drop off as X increases. As X increases to about 90 mm the undesired features vanish into the noise.In addition, the increasing intensity of the benzene peak out to 40 mm indicates an increase in the amount of Raman scattering occurring in the benzene. These effects taken together indicate that most of the Raman scattering detected by the system occurs relatively far from the probe end (in the 10 to 100 mm range). This was a surprising discovery. The intuitive expectation was that most of the scattering would occur within one millimeter of the probe.

The implication of the above discovery is that protecting the probe in with a low loss window will not cause serious degradation in the observed spectra. Subsequent experiments have shown this prediction to be true.

APPENDIX F

EXPERIMENTAL DATA: NUMERICAL APERTURE

Procedure

The measurement of numerical aperture was done using the equipment shown in Figure F-1. Excitation was provided by a HeNe laser. The excited end of the fiber was mounted in a three axis positioner which was, in turn, mounted on a Newport calibrated rotating stage. Geometry was arranged so that the end face of the fiber was at the center of rotation and normal to the axis of the excitation beam when the rotating stage was set to zero. The fiber was coupled to the excitation by directly immersing the end of the fiber into the collimated laser beam. The other end of the fiber was fed directly into the sensor of a Newport 815 power meter. No coupling optics were used at either end of the fiber.

Power meter readings were taken at one degree intervals as the face of the fiber was rotated through the collimated laser beam. Numerical aperture was found by taking the sine of half the contained angle between the 5% power points. Five different fibers were measured using this procedure.

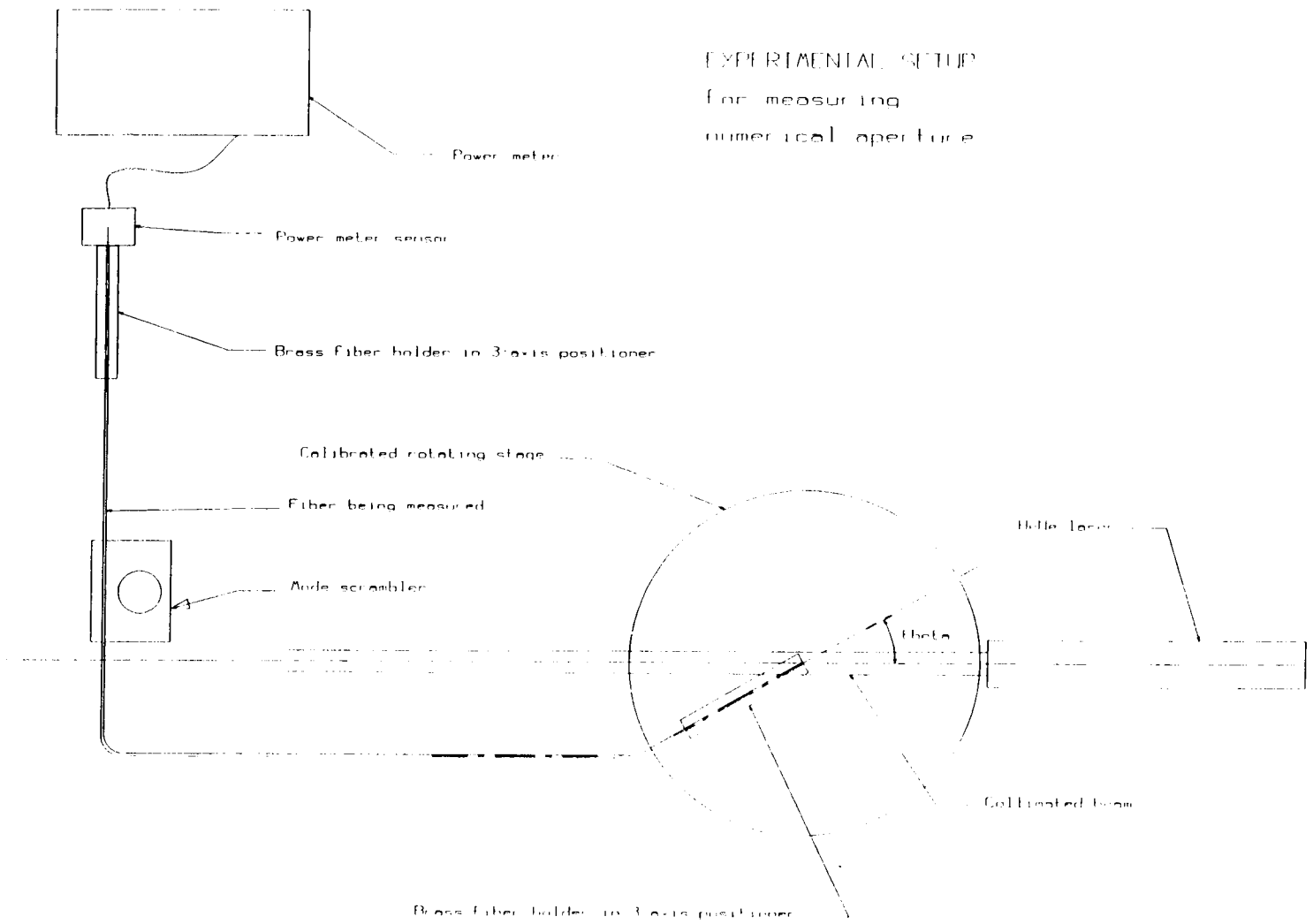


Figure F-1. Numerical Aperture Measurement

Observed Data

The first measurement was taken on a 500 meter roll of Newport F-MLD fiber. The vendor claimed that the core diameter was 101 microns, cladding diameter was 140 microns, and numerical aperture was 0.29. The second set of data was taken on the two meter long excitation (center) fiber of the original 6-around-1 probe loaned to the project by Dow. The core diameter was 400 microns and the expected numerical aperture was 0.22. These data are shown in Table F-1, and in Figures F-2 and F-3.

Similar measurements were made several factory terminated low water 400 micron core fibers. Data were taken for a one meter long C Technologies SRA-4-1-1-02 jumper, and for the excitation fibers of 35 meter and 10 meter fiber bundles custom manufactured by C Technologies. The 35 meter bundle was measured in its coiled configuration as delivered from the factory. The 10 meter fiber was measured in its coiled configuration as delivered from the factory. Since it was in a smaller diameter coil than the 35 meter bundle, it was rewound to about the same coil diameter as the 35 meter bundle and measured again. The data are shown in Table F-2 and Figure F-4.

Table F-1. Newport and Dow Fibers

Stage Setting Degrees	Rotation Angle Degrees	500 Meter Newport Fiber: Power Reading	500 Meter Newport Fiber: % of Peak	2 Meter Dow Probe Fiber: Power Reading	2 Meter Dow Probe Fiber: % of Peak
342	-18	0.565	2.9%	0.0	0.0%
343	-17	2.510	13.1%	0.0	0.0%
344	-16	4.375	22.8%	0.0	0.0%
345	-15	6.430	33.5%	0.0	0.0%
346	-14	8.705	45.4%	1.0	0.2%
347	-13	10.235	53.4%	2.5	0.6%
348	-12	11.640	60.7%	3.5	0.8%
349	-11	12.995	67.8%	8.0	1.8%
350	-10	14.280	74.5%	21.0	4.8%
351	-9	15.615	81.5%	37.0	8.5%
352	-8	16.300	85.0%	53.0	12.2%
353	-7	17.060	89.0%	98.0	22.6%
354	-6	17.850	93.1%	156.0	35.9%
355	-5	18.290	95.4%	263.0	60.6%
356	-4	18.710	97.6%	343.0	79.0%
357	-3	19.140	99.8%	414.0	95.4%
358	-2	19.020	99.2%	429.0	98.8%
359	-1	19.170	100.0%	434.0	100.0%
0	0	18.765	97.9%	430.0	99.1%
1	1	18.245	95.2%	419.5	96.7%
2	2	17.805	92.9%	417.5	96.2%
3	3	17.120	89.3%	423.5	97.6%
4	4	16.220	84.6%	420.5	96.9%
5	5	15.105	78.8%	426.0	98.2%
6	6	14.195	74.0%	427.5	98.5%
7	7	12.735	66.4%	429.5	99.0%
8	8	11.415	59.5%	419.0	96.5%
9	9	9.840	51.3%	395.0	91.0%
10	10	7.895	41.2%	334.5	77.1%
11	11	6.005	31.3%	246.5	56.8%
12	12	4.785	25.0%	168.5	38.8%
13	13	2.855	14.9%	102.5	23.6%
14	14	1.460	7.6%	65.0	15.0%
15	15	0.390	2.0%	36.0	8.3%
16	16	0.045	0.2%	21.5	5.0%
17	17	0.010	0.1%	8.5	2.0%
18	18	0.000	0.0%	5.0	1.2%

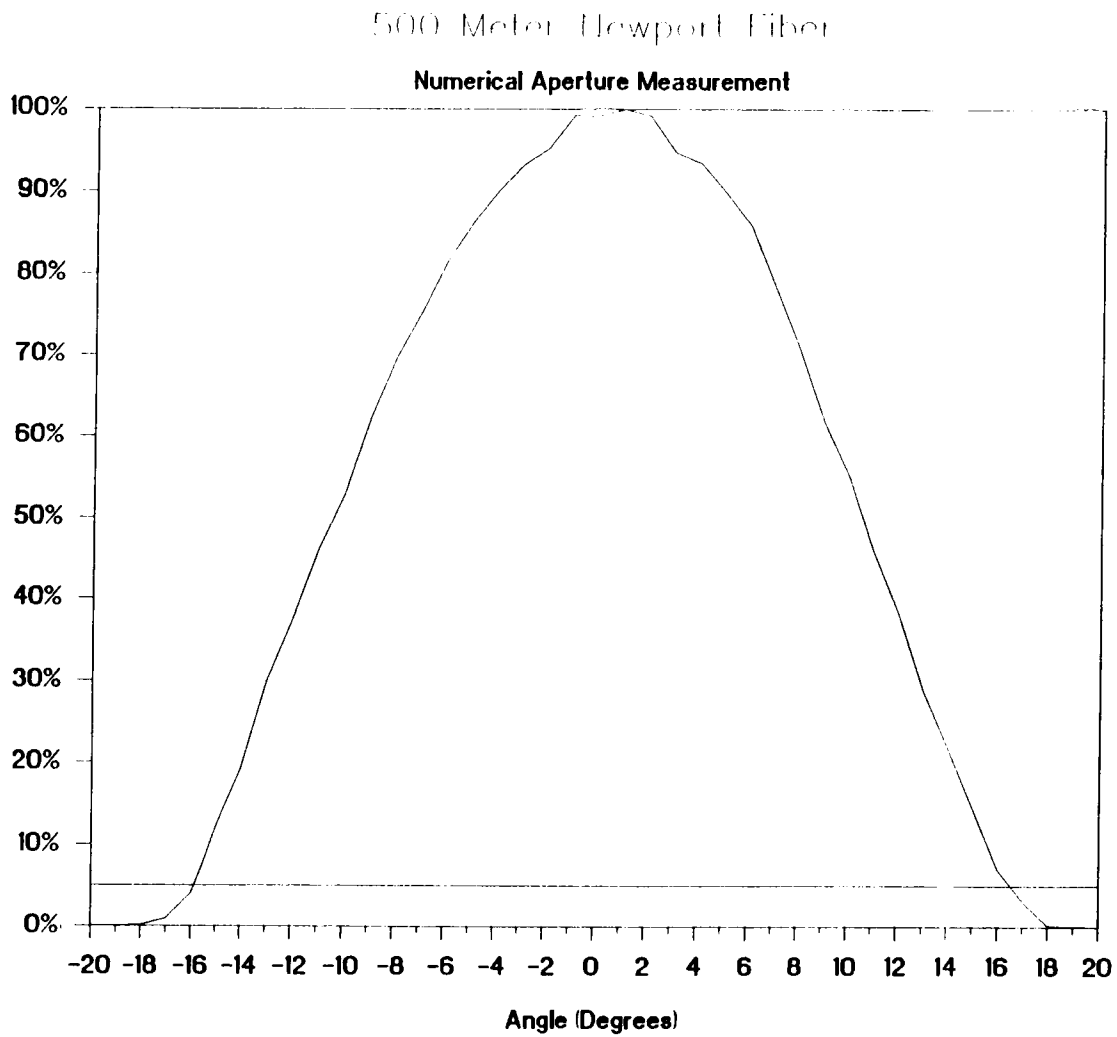


Figure F-2. Newport Fiber Numerical Aperture

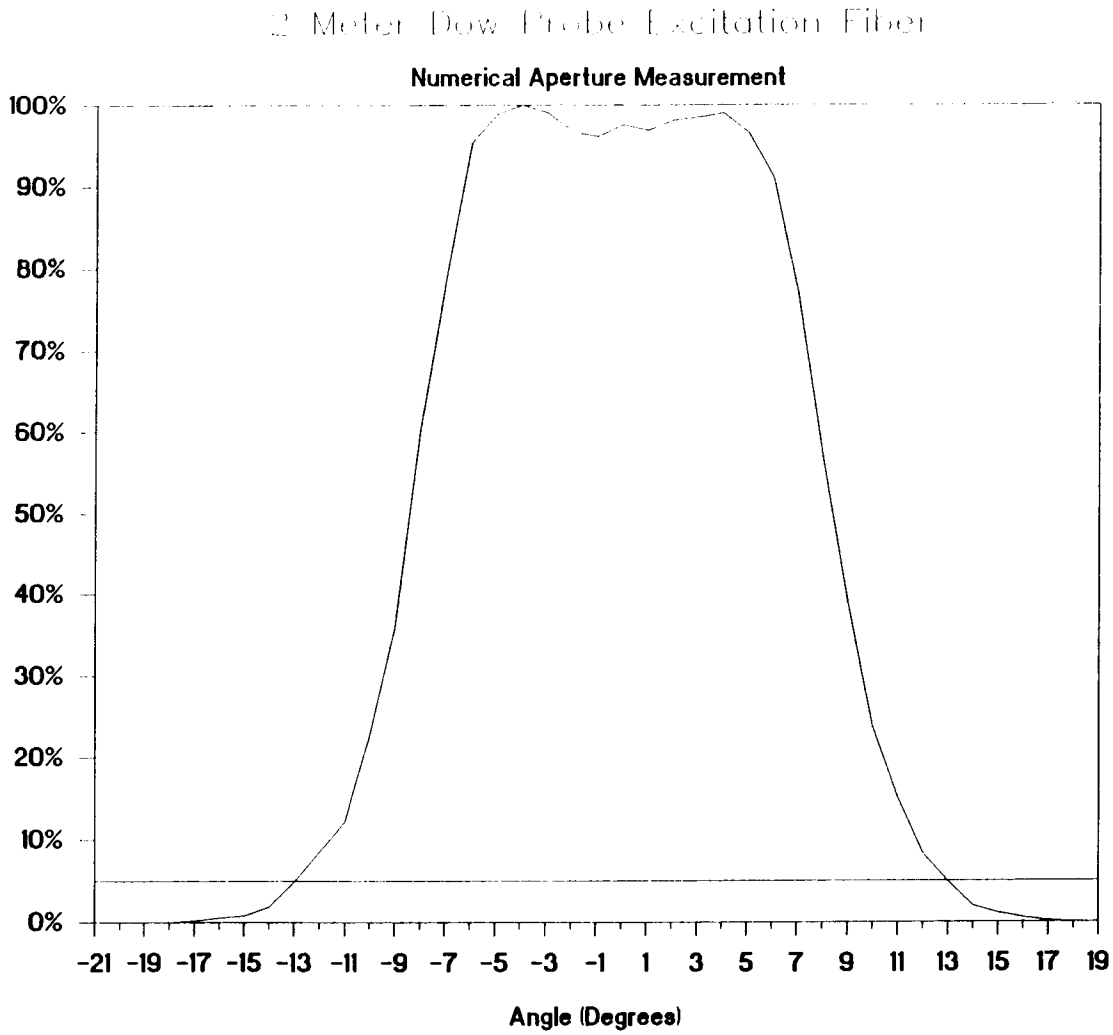


Figure F-3. Dow Probe Numerical Aperture

Table F-2. Low Water Fibers

Stage Setting Degrees	Rotation Angle Degrees	35 Meter Fiber: Power Reading	% peak	10 Meter Fiber: Tight Coil Power Reading	% peak	10 Meter Fiber: Loose Coil Power Reading	% peak	1 Meter Fiber: Power Reading	% peak
40	42	0.000	0.00%	0.000	0.00%	0.000	0.00%	0.000	0.00%
38	40	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.000	0.00%
36	38	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.000	0.00%
34	36	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.001	0.12%
32	34	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.001	0.12%
30	32	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.002	0.24%
29	31	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.003	0.35%
28	30	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.004	0.47%
27	29	0.002	0.31%	0.000	0.00%	0.000	0.00%	0.005	0.59%
26	28	0.003	0.46%	0.001	0.12%	0.001	0.12%	0.009	1.06%
25	27	0.005	0.77%	0.003	0.36%	0.003	0.35%	0.013	1.53%
24	26	0.007	1.08%	0.005	0.59%	0.006	0.70%	0.017	2.00%
23	25	0.009	1.39%	0.007	0.83%	0.007	0.82%	0.023	2.70%
22	24	0.015	2.32%	0.009	1.07%	0.010	1.17%	0.035	4.11%
21	23	0.020	3.09%	0.014	1.66%	0.016	1.87%	0.045	5.29%
20	22	0.025	3.86%	0.018	2.13%	0.020	2.34%	0.058	6.82%
19	21	0.033	5.10%	0.025	2.96%	0.028	3.27%	0.077	9.05%
18	20	0.038	5.87%	0.037	4.38%	0.047	5.49%	0.098	11.52%
17	19	0.051	7.88%	0.055	6.52%	0.062	7.24%	0.122	14.34%
16	18	0.068	10.51%	0.085	10.07%	0.096	11.21%	0.148	17.39%
15	17	0.086	13.29%	0.109	12.91%	0.135	15.77%	0.187	21.97%
14	16	0.111	17.16%	0.157	18.60%	0.190	22.20%	0.235	27.61%
13	15	0.156	24.11%	0.205	24.29%	0.266	31.07%	0.338	39.72%
12	14	0.229	35.39%	0.340	40.28%	0.451	52.69%	0.533	62.63%
11	13	0.303	46.83%	0.549	65.05%	0.703	82.13%	0.685	80.49%
10	12	0.542	83.77%	0.666	78.91%	0.719	84.00%	0.730	85.78%
9	11	0.556	85.94%	0.686	81.28%	0.726	84.81%	0.748	87.90%

Table F-2. (continued)

Stage Setting Degrees	Rotation Angle Degrees	35 Meter Fiber: Power Reading	% peak	10 Meter Fiber: Tight Coil Power Reading	% peak	10 Meter Fiber: Loose Coil Power Reading	% peak	1 Meter Fiber: Power Reading	% peak
8	10	0.562	86.86%	0.699	82.82%	0.734	85.75%	0.762	89.54%
7	9	0.566	87.48%	0.705	83.53%	0.742	86.68%	0.776	91.19%
6	8	0.581	89.80%	0.735	87.09%	0.763	89.14%	0.791	92.95%
5	7	0.600	92.74%	0.764	90.52%	0.795	92.87%	0.807	94.83%
4	6	0.619	95.67%	0.790	93.60%	0.815	95.21%	0.814	95.65%
3	5	0.634	97.99%	0.821	97.27%	0.837	97.78%	0.825	96.94%
2	4	0.642	99.23%	0.832	98.58%	0.855	99.88%	0.842	98.94%
1	3	0.644	99.54%	0.834	98.82%	0.853	99.65%	0.845	99.29%
0	2	0.646	99.85%	0.840	99.53%	0.850	99.30%	0.845	99.29%
359	1	0.646	99.85%	0.842	99.76%	0.856	100.00%	0.851	100.00%
358	0	0.647	100.00%	0.844	100.00%	0.850	99.30%	0.851	100.00%
357	-1	0.645	99.69%	0.843	99.88%	0.845	98.71%	0.850	99.88%
356	-2	0.643	99.38%	0.836	99.05%	0.840	98.13%	0.847	99.53%
355	-3	0.640	98.92%	0.818	96.92%	0.820	95.79%	0.840	98.71%
354	-4	0.637	98.45%	0.786	93.13%	0.795	92.87%	0.830	97.53%
353	-5	0.614	94.90%	0.760	90.05%	0.759	88.67%	0.820	96.36%
352	-6	0.594	91.81%	0.742	87.91%	0.741	86.57%	0.810	95.18%
351	-7	0.578	89.34%	0.736	87.20%	0.733	85.63%	0.798	93.77%
350	-8	0.565	87.33%	0.728	86.26%	0.722	84.35%	0.784	92.13%
349	-9	0.559	86.40%	0.716	84.83%	0.708	82.71%	0.769	90.36%
348	-10	0.552	85.32%	0.625	74.05%	0.692	80.84%	0.753	88.48%
347	-11	0.526	81.30%	0.353	41.82%	0.447	52.22%	0.739	86.84%
346	-12	0.316	48.84%	0.240	28.44%	0.277	32.36%	0.722	84.84%
345	-13	0.191	29.52%	0.168	19.91%	0.189	22.08%	0.668	78.50%
344	-14	0.145	22.41%	0.123	14.57%	0.144	16.82%	0.423	49.71%
343	-15	0.105	16.23%	0.095	11.26%	0.101	11.80%	0.316	37.13%
342	-16	0.082	12.67%	0.069	8.18%	0.069	8.06%	0.232	27.26%
341	-17	0.063	9.74%	0.046	5.45%	0.052	6.07%	0.189	22.21%
340	-18	0.049	7.57%	0.034	4.03%	0.033	3.86%	0.147	17.27%

Table F-2. (continued)

Stage Setting Degrees	Rotation Angle Degrees	35 Meter Fiber: Power Reading	% peak	10 Meter Fiber: Tight Coil Power Reading	% peak	10 Meter Fiber: Loose Coil Power Reading	% peak	1 Meter Fiber: Power Reading	% peak
339	-19	0.037	5.72%	0.025	2.96%	0.021	2.45%	0.110	12.93%
338	-20	0.031	4.79%	0.021	2.49%	0.016	1.87%	0.093	10.93%
337	-21	0.024	3.71%	0.012	1.42%	0.010	1.17%	0.076	8.93%
336	-22	0.019	2.94%	0.009	1.07%	0.007	0.82%	0.059	6.93%
335	-23	0.014	2.16%	0.007	0.83%	0.005	0.58%	0.046	5.41%
334	-24	0.008	1.24%	0.004	0.47%	0.003	0.35%	0.033	3.88%
333	-25	0.006	0.93%	0.002	0.24%	0.001	0.12%	0.024	2.82%
331	-27	0.002	0.31%	0.001	0.12%	0.000	0.00%	0.017	2.00%
330	-28	0.002	0.31%	0.000	0.00%	0.000	0.00%	0.015	1.76%
329	-29	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.009	1.06%
328	-30	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.005	0.59%
326	-32	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.003	0.35%
324	-34	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.002	0.24%
322	-36	0.001	0.15%	0.000	0.00%	0.000	0.00%	0.001	0.12%
320	-38	0.000	0.00%	0.000	0.00%	0.000	0.00%	0.001	0.12%

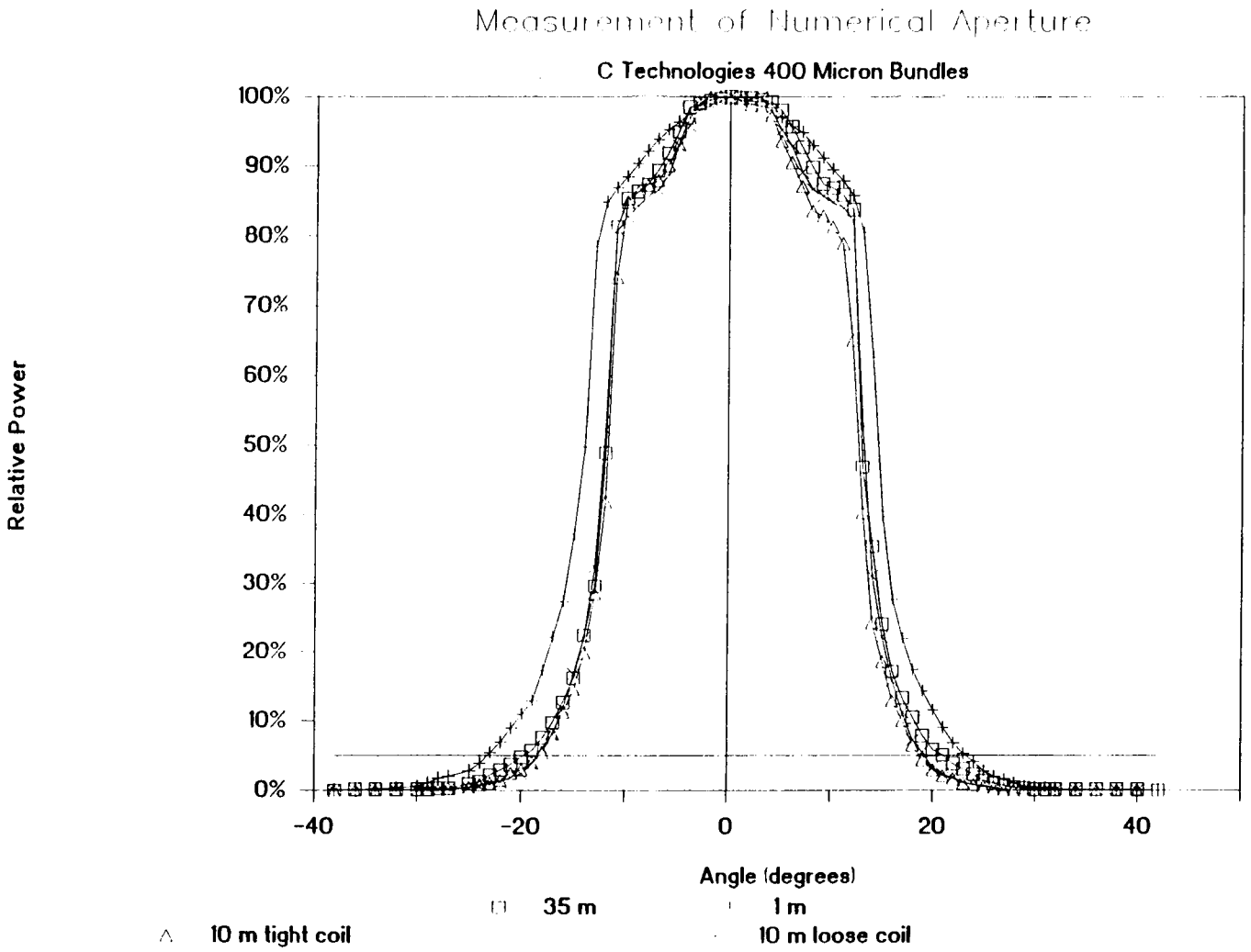


Figure F-4. Low Water Numerical Apertures

Conclusions

Fractional angles corresponding to 5% of peak intensity are calculated by linear interpolation. The observed numerical apertures are summarized in Table F-3.

Table F-3. Numerical Apertures

Fiber Type	Expected Value	Observed Value
Newport 500 m	.29	.278
Dow 2 m	.22	.225
C 1 m	.22	.395
C 10 m tight coil	.22	.318
C 10 m loose coil	.22	.323
C 35 m	.22	.341

It was anticipated that long fibers would have a lower numerical aperture than the short fibers due to attenuation of complex modes. It was surprising that the 10 meter fiber had a slightly smaller numerical aperture than the 35 meter fiber. It is suspected that a tight local radius somewhere within the 10 meter bundle caused it to attenuate complex modes a little more than the 35 meter bundle.

Figure F-4 shows a distinct head and shoulders pattern in the low water fiber response. This is plainly visible in visible light when the output of the fiber is projected onto a white card. The head and shoulders pattern is also visible in the infrared when the fiber output is projected onto a fluorescent card.

The surprising result of these measurements is that the numerical aperture of the low water fibers appears to be in the neighborhood of .33 instead of the expected value of .22.

APPENDIX G

EXPERIMENTAL DATA: CONNECTION LOSS MEASUREMENT

Procedure

This experiment consisted of coupling the 1.064 micron output of the CVI C95 laser into 1 meter lengths of 400 micron core diameter fibers terminated with low loss connectors and connected with bushings. The fibers were C Technologies SRA-4-1-1-02 fibers with factory installed connectors. The fibers were joined with C Technologies bushings. Coupling of the laser to the fiber was by a 5X microscope objective (0.12 numerical aperture). The other end of the fiber was pointed directly into the sensor of a Coherent Model 210 optical power meter.

The ends of each of the three different fibers were tagged with numbers so that the losses of all the different possible combinations of faces could be measured. First the fiber receiving power directly from the laser was adjusted in the 3 axis positioner until maximum output was read at the other end. Then the other fiber(s) to be tested were connected via bushings, and the resulting output was measured. Then the other fibers were disconnected, and the output of the fiber directly coupled to the laser was measured again. The average of the first and third power readings was taken as the input power to the connection(s). This was to compensate for the small but noticeable power

drift of the laser. The middle reading was the output reading. The ratio of output to input powers was the basis for calculating the dB loss for the system of connectors. 36 different combinations of sequences of tagged ends were measured by the above procedure. Each set of measurements was repeated four different times.

Observed Data

Tables G-1 and G-2 and Figures G-1 and G-2 are a statistical summary of the observations. These summaries are based on the individual observations listed in Tables G-3, G-4, G-5, and G-6.

It is obvious from the data that run 4 is out of line with all the others. Close inspection of the raw data suggests that end 4 of the 3-4 fiber was damaged when the 3-4:5-6 reading was attempted on run 4. The first reading was 570 mw and the third was 360 mw. It is also noteworthy that measurements in the fourth run not involving fiber 3-4 are in line with previous runs. Inspection of end 4 with a lens while shining white light through end 3 indicates visible damage to end 4. Inspection of all ends under the microscope indicates considerable scoring and scraping of the finish. Note that these fibers were brand new at the beginning of the experiment.

Table G-1. Loss Statistics per Measurement

Losses: dB per Connection				
	Run 1	Run 2	Run 3	Run 4
Count:	36	36	36	36
Mean:	0.81669	0.88373	0.9643	1.9613
Std Dev:	0.14549	0.15955	0.1293	0.7941
Minimum:	0.49882	0.20203	0.6835	0.8266
Maximum:	1.11633	1.10631	1.1760	3.6797

Table G-2. Cumulative Loss Statistics

Losses: dB per Connection				
	Run 1	Run 1-2	Run 1-3	Run 1-4
Count:	36	72	108	144
Mean:	0.81669	0.85021	0.8882	1.1565
Std Dev:	0.14549	0.15632	0.1573	0.6262
Minimum:	0.49882	0.20203	0.2020	0.2020
Maximum:	1.11633	1.11633	1.1760	3.6797

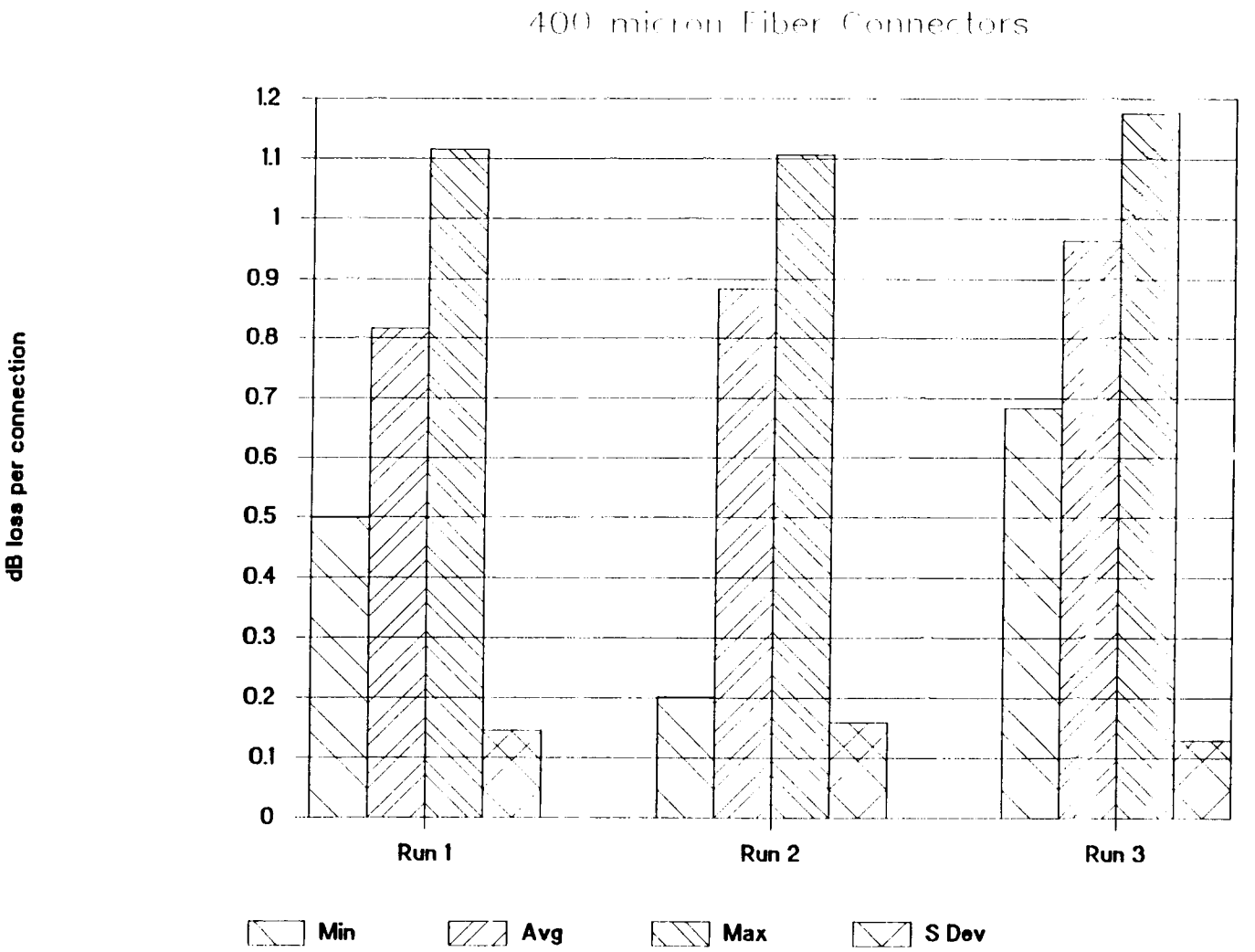


Figure G-1. Attenuation Statistics per Measurement

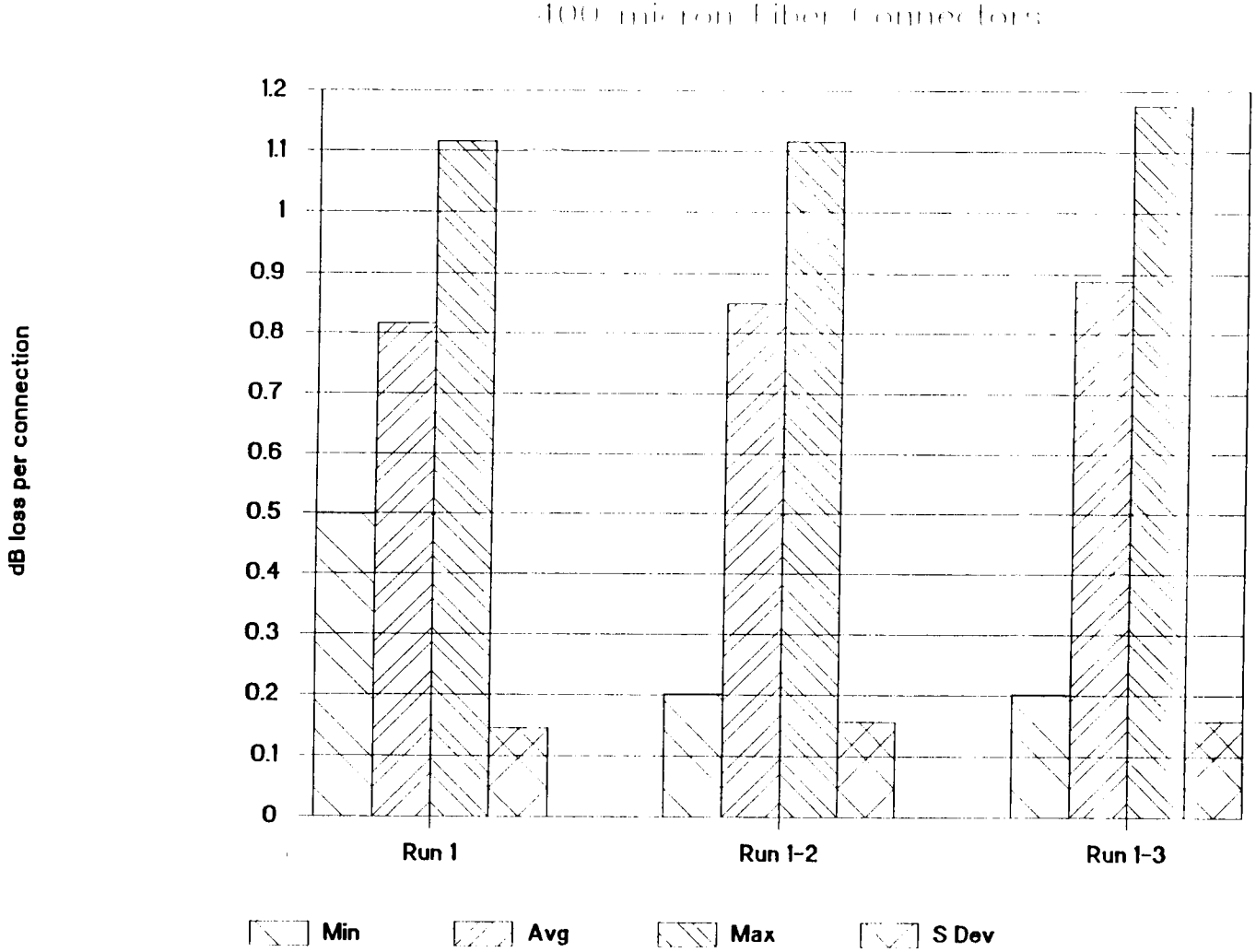


Figure G-2. Cumulative Attenuation Statistics

Table G-3. First Connector Loss Measurement

Connector Sequence	Input Power (Milliwatts) Beginning	End	Output Power (mw)	Average Input (mw)	Ratio	dB	dB/connector
1-2 3-4 5-6	585	582	402	583.5	0.6889	-1.618	-0.80907
1-2 3-4 6-5	582	585	440	583.5	0.7540	-1.225	-0.61294
1-2 4-3 5-6	570	580	405	575.0	0.7043	-1.522	-0.76106
1-2 4-3 6-5	570	575	455	572.5	0.7947	-0.997	-0.49882
2-1 3-4 5-6	565	565	360	565.0	0.6371	-1.957	-0.97872
2-1 3-4 6-5	565	565	410	565.0	0.7256	-1.392	-0.69632
2-1 4-3 5-6	560	560	375	560.0	0.6696	-1.741	-0.87078
2-1 4-3 6-5	550	560	415	555.0	0.7477	-1.262	-0.63122
1-2 5-6 3-4	575	575	450	575.0	0.7826	-1.064	-0.53227
1-2 6-5 3-4	560	565	415	562.5	0.7377	-1.320	-0.66037
1-2 5-6 4-3	560	560	360	560.0	0.6428	-1.918	-0.95942
1-2 6-5 4-3	560	570	370	565.0	0.6548	-1.838	-0.91923
2-1 5-6 3-4	550	565	400	557.5	0.7174	-1.441	-0.72092
2-1 6-5 3-4	570	550	365	560.0	0.6517	-1.858	-0.92947
2-1 5-6 4-3	540	550	345	545.0	0.6330	-1.985	-0.99288
2-1 6-5 4-3	550	550	345	550.0	0.6272	-2.025	-1.01271
5-6 1-2 3-4	510	510	355	510.0	0.6960	-1.573	-0.78670
6-5 1-2 3-4	525	530	355	527.5	0.6729	-1.719	-0.85997
5-6 1-2 4-3	510	515	370	512.5	0.7219	-1.414	-0.70746
6-5 1-2 4-3	530	530	355	530.0	0.6698	-1.740	-0.87023
5-6 2-1 3-4	520	515	365	517.5	0.7053	-1.516	-0.75808
6-5 2-1 3-4	525	520	365	522.5	0.6985	-1.557	-0.77896
5-6 2-1 4-3	515	510	325	512.5	0.6341	-1.978	-0.98905
6-5 2-1 4-3	515	525	335	520.0	0.6442	-1.909	-0.95479
1-2 3-4	582	582	502	582.0	0.8625	-0.642	-0.64219
1-2 4-3	582	582	500	582.0	0.8591	-0.659	-0.65952
2-1 3-4	560	560	470	560.0	0.8392	-0.760	-0.76090
2-1 4-3	550	555	460	552.5	0.8325	-0.795	-0.79574
3-4 5-6	550	550	455	550.0	0.8272	-0.823	-0.82351
3-4 6-5	550	550	465	550.0	0.8454	-0.729	-0.72909
4-3 5-6	560	550	450	555.0	0.8108	-0.910	-0.91080
4-3 6-5	540	565	455	552.5	0.8235	-0.843	-0.84320
1-2 5-6	565	565	455	565.0	0.8053	-0.940	-0.94037
1-2 6-5	575	575	475	575.0	0.8260	-0.829	-0.82974
2-1 5-6	555	555	435	555.0	0.7837	-1.058	-1.05803
2-1 6-5	565	560	435	562.5	0.7733	-1.116	-1.11633

Table G-4. Second Connector Loss Measurement

Connector Sequence	Input Power (Milliwatts) Beginning	End	Output Power (mw)	Average Input (mw)	Ratio	dB	dB/connector
1-2 3-4 5-6	540	540	345	540.0	0.6388	-1.945	-0.97287
1-2 3-4 6-5	540	535	365	537.5	0.6790	-1.680	-0.84042
1-2 4-3 5-6	535	545	355	540.0	0.6574	-1.821	-0.91082
1-2 4-3 6-5	540	545	355	542.5	0.6543	-1.841	-0.92085
2-1 3-4 5-6	540	545	335	542.5	0.6175	-2.093	-1.04677
2-1 3-4 6-5	540	540	340	540.0	0.6296	-2.009	-1.00457
2-1 4-3 5-6	540	540	355	540.0	0.6574	-1.821	-0.91082
2-1 4-3 6-5	545	545	350	545.0	0.6422	-1.923	-0.96164
1-2 5-6 3-4	545	545	365	545.0	0.6697	-1.741	-0.87051
1-2 6-5 3-4	540	540	345	540.0	0.6388	-1.945	-0.97287
1-2 5-6 4-3	535	530	355	532.5	0.6666	-1.760	-0.88045
1-2 6-5 4-3	520	540	345	530.0	0.6509	-1.864	-0.93228
2-1 5-6 3-4	545	545	340	545.0	0.6238	-2.049	-1.02458
2-1 6-5 3-4	545	540	350	542.5	0.6451	-1.903	-0.95165
2-1 5-6 4-3	540	540	350	540.0	0.6481	-1.883	-0.94162
2-1 6-5 4-3	545	550	360	547.5	0.6575	-1.820	-0.91040
5-6 1-2 3-4	540	540	350	540.0	0.6481	-1.883	-0.94162
6-5 1-2 3-4	540	540	340	540.0	0.6296	-2.009	-1.00457
5-6 1-2 4-3	540	535	350	537.5	0.6511	-1.863	-0.93155
6-5 1-2 4-3	540	540	340	540.0	0.6296	-2.009	-1.00457
5-6 2-1 3-4	535	540	330	537.5	0.6139	-2.118	-1.05932
6-5 2-1 3-4	540	535	335	537.5	0.6232	-2.053	-1.02666
5-6 2-1 4-3	540	535	370	537.5	0.6883	-1.621	-0.81088
6-5 2-1 4-3	535	535	350	535.0	0.6542	-1.842	-0.92142
1-2 3-4	540	540	460	540.0	0.8518	-0.696	-0.69635
1-2 4-3	545	540	465	542.5	0.8571	-0.669	-0.66946
2-1 3-4	555	545	525	550.0	0.9545	-0.202	-0.20203
2-1 4-3	545	525	455	535.0	0.8504	-0.703	-0.70342
3-4 5-6	520	525	405	522.5	0.7751	-1.106	-1.10631
3-4 6-5	525	525	430	525.0	0.8190	-0.866	-0.86690
4-3 5-6	525	525	420	525.0	0.8	-0.969	-0.96910
4-3 6-5	525	520	445	522.5	0.8516	-0.697	-0.69726
1-2 5-6	540	540	455	540.0	0.8425	-0.743	-0.74382
1-2 6-5	540	545	460	542.5	0.8479	-0.716	-0.71641
2-1 5-6	540	540	440	540.0	0.8148	-0.889	-0.88941
2-1 6-5	530	540	445	535.0	0.8317	-0.799	-0.79993

Table G-5. Third Connector Loss Measurement

Connector Sequence	Input Power (Milliwatts) Beginning	End	Output Power (mw)	Average Input (mw)	Ratio	dB	dB/connector
1-2 3-4 5-6	525	530	320	527.5	0.6066	-2.170	-1.08536
1-2 3-4 6-5	530	540	335	535.0	0.6261	-2.033	-1.01654
1-2 4-3 5-6	540	540	325	540.0	0.6018	-2.205	-1.10255
1-2 4-3 6-5	540	540	360	540.0	0.6666	-1.760	-0.88045
2-1 3-4 5-6	550	555	400	552.5	0.7239	-1.402	-0.70136
2-1 3-4 6-5	555	550	365	552.5	0.6606	-1.800	-0.90019
2-1 4-3 5-6	550	555	335	552.5	0.6063	-2.172	-1.08643
2-1 4-3 6-5	550	560	360	555.0	0.6486	-1.879	-0.93995
1-2 5-6 3-4	540	535	365	537.5	0.6790	-1.680	-0.84042
1-2 6-5 3-4	535	535	315	535.0	0.5887	-2.300	-1.15021
1-2 5-6 4-3	535	535	340	535.0	0.6355	-1.968	-0.98437
1-2 6-5 4-3	540	540	340	540.0	0.6296	-2.009	-1.00457
2-1 5-6 3-4	560	560	350	560.0	0.625	-2.041	-1.02059
2-1 6-5 3-4	550	550	320	550.0	0.5818	-2.352	-1.17606
2-1 5-6 4-3	550	550	330	550.0	0.6	-2.218	-1.10924
2-1 6-5 4-3	550	550	320	550.0	0.5818	-2.352	-1.17606
5-6 1-2 3-4	550	550	340	550.0	0.6181	-2.088	-1.04441
6-5 1-2 3-4	570	570	360	570.0	0.6315	-1.995	-0.99786
5-6 1-2 4-3	540	550	355	545.0	0.6513	-1.861	-0.93084
6-5 1-2 4-3	570	570	365	570.0	0.6403	-1.935	-0.96790
5-6 2-1 3-4	550	560	360	555.0	0.6486	-1.879	-0.93995
6-5 2-1 3-4	570	570	350	570.0	0.6140	-2.118	-1.05903
5-6 2-1 4-3	560	560	340	560.0	0.6071	-2.167	-1.08354
6-5 2-1 4-3	570	570	345	570.0	0.6052	-2.180	-1.09027
1-2 3-4	540	540	430	540.0	0.7962	-0.989	-0.98925
1-2 4-3	540	540	460	540.0	0.8518	-0.696	-0.69635
2-1 3-4	555	555	440	555.0	0.7927	-1.008	-1.00840
2-1 4-3	550	550	460	550.0	0.8363	-0.776	-0.77604
3-4 5-6	510	515	420	512.5	0.8195	-0.864	-0.86444
3-4 6-5	515	515	440	515.0	0.8543	-0.683	-0.68354
4-3 5-6	530	530	430	530.0	0.8113	-0.908	-0.90807
4-3 6-5	530	530	440	530.0	0.8301	-0.808	-0.80823
1-2 5-6	540	540	430	540.0	0.7962	-0.989	-0.98925
1-2 6-5	540	540	445	540.0	0.8240	-0.840	-0.84033
2-1 5-6	550	560	440	555.0	0.7927	-1.008	-1.00840
2-1 6-5	560	560	460	560.0	0.8214	-0.854	-0.85430

Table G-6. Fourth Connector Loss Measurement

Connector Sequence	Input Power (Milliwatts) Beginning	End	Output Power (mw)	Average Input (mw)	Ratio	dB	dB/connector
1-2 3-4 5-6	570	570	190	570.0	0.3333	-4.771	-2.38560
1-2 3-4 6-5	565	560	195	562.5	0.3466	-4.600	-2.30043
1-2 4-3 5-6	565	565	170	565.0	0.3008	-5.215	-2.60799
1-2 4-3 6-5	565	570	195	567.5	0.3436	-4.639	-2.31965
2-1 3-4 5-6	570	560	175	565.0	0.3097	-5.090	-2.54505
2-1 3-4 6-5	565	560	190	562.5	0.3377	-4.713	-2.35684
2-1 4-3 5-6	560	555	165	557.5	0.2959	-5.287	-2.64380
2-1 4-3 6-5	560	565	195	562.5	0.3466	-4.600	-2.30043
1-2 5-6 3-4	570	565	220	567.5	0.3876	-4.115	-2.05771
1-2 6-5 3-4	570	555	220	562.5	0.3911	-4.076	-2.03849
1-2 5-6 4-3	560	560	175	560.0	0.3125	-5.051	-2.52574
1-2 6-5 4-3	565	565	175	565.0	0.3097	-5.090	-2.54505
2-1 5-6 3-4	560	565	205	562.5	0.3644	-4.383	-2.19184
2-1 6-5 3-4	565	565	210	565.0	0.3716	-4.298	-2.14914
2-1 5-6 4-3	565	570	170	567.5	0.2995	-5.235	-2.61758
2-1 6-5 4-3	565	555	170	560.0	0.3035	-5.177	-2.58869
5-6 1-2 3-4	515	520	330	517.5	0.6376	-1.953	-0.97698
6-5 1-2 3-4	510	515	315	512.5	0.6146	-2.113	-1.05691
5-6 1-2 4-3	510	520	335	515.0	0.6504	-1.867	-0.93381
6-5 1-2 4-3	540	540	335	540.0	0.6203	-2.073	-1.03674
5-6 2-1 3-4	510	515	325	512.5	0.6341	-1.978	-0.98905
6-5 2-1 3-4	545	540	320	542.5	0.5898	-2.292	-1.14624
5-6 2-1 4-3	515	510	335	512.5	0.6536	-1.846	-0.92324
6-5 2-1 4-3	535	540	340	537.5	0.6325	-1.988	-0.99449
1-2 3-4	555	560	290	557.5	0.5201	-2.838	-2.83846
1-2 4-3	560	560	240	560.0	0.4285	-3.679	-3.67976
2-1 3-4	560	555	300	557.5	0.5381	-2.691	-2.69123
2-1 4-3	555	560	240	557.5	0.4304	-3.660	-3.66033
3-4 5-6	360	360	210	360.0	0.5833	-2.340	-2.34083
3-4 6-5	360	365	210	362.5	0.5793	-2.370	-2.37088
4-3 5-6	390	390	275	390.0	0.7051	-1.517	-1.51731
4-3 6-5	390	385	305	387.5	0.7870	-1.039	-1.03971
1-2 5-6	560	560	435	560.0	0.7767	-1.096	-1.09698
1-2 6-5	560	565	465	562.5	0.8266	-0.826	-0.82669
2-1 5-6	555	555	4[illegible]	555.0	0.7387	-1.315	-1.31509
2-1 6-5	560	560	44[illegible]	560.0	0.7946	-0.998	-0.99828

Conclusions

Due to damage to end 4, the fourth run is not considered valid. Loss statistics for design purposes should be taken from the composite results of the first three runs. An average loss of 0.9 dB per connection is a reasonable assumption based on the observed data.

Note that there is a degradation of performance and appearance of the connectors with handling. Each run produced about 0.1 dB worse performance than the previous run. During each run, each connector was connected and disconnected between 50 and 100 times. In practice, connections and disconnections should be made a bare minimum number of times.

APPENDIX H

CODE LISTING: PROBE ANALYSIS PROGRAM

```
/*UT Measurement and Control Engineering Center                    */
/*                                                                 */
/* M J Roberts, S W Kercel                                         */
/*                                                                 */
/* probe.c Version 0   January 16, 1990                            */
/*                                                                 */
/* WARNING AND DISCLAIMER:                                         */
/* This program is in the public domain and is offered free        */
/* of charge on an AS IS basis. It has been checked with           */
/* representative data, and it returns results consistent          */
/* with experimental observations.                                 */
/*                                                                 */
/* HOWEVER,                                                        */
/*                                                                 */
/* the authors make no guarantees concerning its performance       */
/* Neither the authors nor The University of Tennessee             */
/* assume any liability for damages that may result from           */
/* the use of this program.                                        */
/*                                                                 */
/*                                                                 */
/* Description of Program:                                         */
/* This is a VAX-VMS C implementation of a numerical eval-         */
/* uation of the multiple integral that predicts the               */
/* received Raman power of two fibers in an angled fiber           */
/* optic probe.                                                    */
/*                                                                 */
/* This program obtains probe parameters interactively from        */
/* the user. Then it computes the value of the integral.           */
/* Finally, numerical results are written to a file.               */
/*                                                                 */
/* Input Data (Supplied interactively by user):                    */
/*                                                                 */
/* Core diameter of excitation fiber                               */
/* Cladding diameter of excitation fiber                           */
/* Core diameter of receiving fiber                                */
/* Cladding diameter of receiving fiber                            */
/* Spacing between fiber ends                                      */
/* External critical angle of excitation fiber                     */
/* External critical angle of receiving fiber                      */
/* Angle between fiber optical axes                                */
/* Raman scattering cross section                                  */
/* Number density of scatterers                                    */
/* Space constant of attenuation                                   */
/* Analysis depth along exciting fiber axis                        */
/* Axial offset of receiving fiber                                 */
/*                                                                 */
/* Output Data (written to probe.dat)                              */
/*                                                                 */
/* received Raman power as a function of the above data            */
/* Raman power as a function of distance along optical axis        */
/*                                                                 */
/* Functions called from main():                                   */
/*                                                                 */
/* headline - prints friendly greeting to user                     */
/* inp_data - gets probe parameters from user                      */
```

```
                /* error_trap1 - tests ranges of input data                  */
                /* error_trap2 - detects illegal combinations of parameters */
                /* fig_power - evaluates the integral                        */
                /* results - stores output data to file                      */
                /*                                                           */
                /*************************************************************/

                /*compiler directives*/

                # include <stdio.h>
                # include <math.h>
                # define TRUE 1
                # define FALSE 0
                # define NUMPARMS 13
                # define STRLEN 70
                # define RADIAL 4 /*number of finite elements in radial dimension*/
                # define AZIMUTH 16 /*number of finite elements in azimuthal dim*/
                # define AXIAL 80 /*number of finite elements in axial dimension*/
                # define PI 3.14159265358979323846
                # define RADIAN 0.01745329251994329591

                /* program actually starts here */

                main()
                {
                        /* declaration and initialization of variables */

                        int mainloop = TRUE; /*TRUE => keep going*/
                        int firstpass = TRUE;/*TRUE => first time thru inp_data*/
                        int file_good; /*TRUE => no problems opening file*/
                        int erflag;/* TRUE => user input data not correct*/
                        double fibarray[NUMPARMS];/*fiber paramaters from user*/
                        double zpower[AXIAL];/*Raman power vs. distance*/
                        double cplg;/*probe received Raman power*/
                        FILE *fp,*fopen();/*file handling pointers*/

                        /* open a file for output data*/

                        fp = fopen("probe.dat","w");

                        /* procede if file opened ok */

                        if (fp != NULL)
                         {
                                headline();
                                while(mainloop)
                                 {
                                  erflag = TRUE;
                                  while(erflag && mainloop)
                                   {
                                    inp_data(fibarray,&mainloop,&firstpass);/*input*/
                                    if (mainloop)
                                     {
                                      erflag = error_trap1(fibarray);
                                      erflag = (erflag || error_trap2(fibarray));
                                     }
```

```
583  4                      }
584  3                     if (mainloop)
585  3                      {
586  4                       fig_power(fibarray,zpower,&cplg);/*process*/
587  4                       results(fibarray,zpower,&cplg,fp);/*output*/
588  4                      }
589  3                    }
590  2                   fclose(fp);/*close data file*/
591  2                 }
592  1                else
593  1                 printf("Output file probe.dat not successfully opened\n");
594  1
595  1       }
596
597           /* Function headline()                                          */
598           /*                                                              */
599           /* Prints initial information                                   */
600           /*                                                              */
601           /* Input data: None                                             */
602           /*                                                              */
603           /* Output data: Opening screen data                             */
604           /*                                                              */
605           /* Functions called: None                                       */
606           /*                                                              */
607           /*--------------------------------------------------------------*/
608
609           headline()
610
611           {
612  1                    printf("\nUniversity of Tennessee");
613  1                    printf("\nMeasurement and Control Engineering Center");
614  1                    printf("\n\nProbe analysis program");
615  1                    printf("\nAlgorithm developed by M J Roberts");
616  1                    printf("\nVAX C implementation by S W Kercel");
617  1                    printf("\nUser supplies probe data interactively\n");
618  1                    printf("\n                                                                ");
619  1                    printf("\n WARNING AND DISCLAIMER:                                       ");
620  1                    printf("\n This program is in the public domain, and is offered free ");
621  1                    printf("\n of charge on an AS IS basis. It has been checked with    ");
622  1                    printf("\n representative data, and it returns results consistent   ");
623  1                    printf("\n with experimental observations.                           ");
624  1                    printf("\n                                                           ");
625  1                    printf("\n HOWEVER,                                                 ");
626  1                    printf("\n                                                           ");
627  1                    printf("\n the authors make no guarantees concerning its performance.");
628  1                    printf("\n Neither the authors nor The University of Tennessee      ");
629  1                    printf("\n assume any liability for damages that may result from    ");
630  1                    printf("\n the use of this program.                                 ");
631  1                    printf("\n                                                           ");
632  1       }
633
634           /* function inp_data()                                          */
635           /*                                                              */
636           /* function to obtain the twelve parameters that describe       */
637           /* a given probe.                                               */
638           /*                                                              */
639           /* On the first pass, the function gets all twelve para-        */
```

```
640          /* meters. On successive passes, it updates parameters      */
641          /* selected by the user. The updated parameter array is     */
642          /* passed back to the calling function.                     */
643          /*                                                          */
644          /* Input data (furnished by user thru get_parms):           */
645          /* Each parameter is an element of fiber[]                  */
646          /*                                                          */
647          /* Output data (passed back to calling function):           */
648          /*  flag - TRUE => keep main loop going                     */
649          /*  passone - TRUE => first time thru inp_data()            */
650          /*  Probe parameters are elements of fiber[], specifically: */
651          /*   fiber[0] - core diameter of excitation fiber (microns) */
652          /*   fiber[1] - cladding dia of excitation fiber (microns)  */
653          /*   fiber[2] - core diameter of receiving fiber (microns)  */
654          /*   fiber[3] - cladding dia of receiving fiber (microns)   */
655          /*   fiber[4] - spacing between fiber ends (microns)        */
656          /*   fiber[5] - ext critical angle of excitation fiber (deg)*/
657          /*   fiber[6] - ext critical angle of receiving fiber (deg) */
658          /*   fiber[7] - angle between fiber optical axes (degrees)  */
659          /*   fiber[8] - Raman scattering cross sect (cm^2/molecule sr)*/
660          /*   fiber[9] - number density of scatterers / cm^3         */
661          /*   fiber[10] - space constant for attenuation (microns)   */
662          /*   fiber[11] - Analysis depth along exciting fiber axis   */
663          /*               (excitation fiber core diameters)          */
664          /*                                                          */
665          /* Functions called:                                        */
666          /*  get_parms - get all 12 parameters                       */
667          /*  get_one - get partameters selected by user              */
668          /*                                                          */
669          /**********************************************************/
670
671          inp_data(fiber,flag,passone)
672          double fiber[];/*fiber parameters furnished by user*/
673          int *flag;/*TRUE => keep main loop going*/
674          int *passone;/*TRUE => first time thru function*/
675
676          {
677     1            /* local declarations and initializations */
678     1            char resp;/* C => continue */
679     1            char s[81];/*input to key buffer*/
680     1            char s1[80];/*dumping ground*/
681     1
682     1            /* query user whether or not to keep going */
683     1            printf("\n");
684     1            printf("(C)ontinue or (Q)uit program? ");
685     1            scanf("%s",s);
686     1            sscanf(s,"%c%s",&resp,s1);
687     1
688     1            if ((resp != 'C') && (resp != 'c'))
689     1                  *flag=FALSE;
690     1            else
691     1             {
692     2               if (*passone)
693     2                {
694     3                     *passone = FALSE;
695     3                     get_parms(fiber);
696     3                 }
```

```
697  2                 else
698  2                      get_one(fiber);
699  2               }
700  1       }
701
702             /* function error_trap1()                                    */
703             /*                                                           */
704             /* Function tests for various range errors                   */
705             /*                                                           */
706             /* Input Data (parameter):                                   */
707             /*   fib[] - array of user supplied data                     */
708             /*                                                           */
709             /* Output Data (returned in call):                           */
710             /*   locflag - TRUE => error condition exists                */
711             /*                                                           */
712             /* Functions called: None                                    */
713             /*                                                           */
714             /*************************************************************/
715
716             error_trap1(fib)
717             double fib[];/*fiber parameters furnished by user*/
718
719             {
720  1                  /*local declarations and initializations*/
721  1                  int locflag = FALSE;/*FALSE=>no errors*/
722  1
723  1                  if (fib[1] < fib[0])
724  1                   {
725  2                    printf("\nExcitation core diameter less than clad diameter");
726  2                    locflag = TRUE;
727  2                   }
728  1                  if (fib[3] < fib[2])
729  1                   {
730  2                    printf("\nReceiving core diameter less than clad diameter");
731  2                    locflag = TRUE;
732  2                   }
733  1                  if ((fib[5] >= 90) || (fib[5] <= 0))
734  1                   {
735  2                    printf("\nExcitation critical angle outside 0<th<90");
736  2                    locflag = TRUE;
737  2                   }
738  1                  if ((fib[6] >= 90) || (fib[6] <= 0))
739  1                   {
740  2                    printf("\nReceiving critical angle outside 0<th<90");
741  2                    locflag = TRUE;
742  2                   }
743  1                  if ((fib[7] >= 90) || (fib[7] < 0))
744  1                   {
745  2                    printf("\nAngle between optical axes outside 0<=th<90");
746  2                    locflag = TRUE;
747  2                   }
748  1                  if (fib[10] < 0)
749  1                   {
750  2                    printf("\nSpace constant less than 0");
751  2                    locflag = TRUE;
752  2                   }
753  1                  return(locflag);
```

```
754  1      }
755
756          /* function error_trap2()                                    */
757          /*                                                           */
758          /* Function checks for the following illegal conditions:     */
759          /*                                                           */
760          /*  Condition 1 - Near corner of receiving fiber             */
761          /*                protrudes into exciting cone               */
762          /*  Condition 2 - Far corner of receiving fiber              */
763          /*                protrudes into exciting cone               */
764          /*  Condition 3 - Near corner of exciting fiber              */
765          /*                protrudes into receiving cone              */
766          /*  Condition 4 - Far corner of exciting fiber               */
767          /*                protrudes into receiving cone              */
768          /*                                                           */
769          /* Input data (parameter):                                   */
770          /*  fib[] - array of user supplied data                      */
771          /*                                                           */
772          /* Output data (returned in call):                           */
773          /*  locflag - TRUE => error condition exists                 */
774          /*                                                           */
775          /* Functions called: None                                    */
776          /*                                                           */
777          /***********************************************************/
778
779          error_trap2(fib)
780          double fib[];/*fiber parameters furnished by user*/
781
782          {
783  1               /* local declarations and initializations */
784  1               int locflag = FALSE;/*FALSE => no error*/
785  1               double d;/*spacing between nearest points of fibers*/
786  1                       /*along axis of symmetry*/
787  1               double zr;/*spacing between nearest points of fibers*/
788  1                        /*along exciting optical axis*/
789  1               double rcle;/*exciting fiber clad radius*/
790  1               double rcoe;/*exciting fiber core radius*/
791  1               double rclr;/*receiving fiber clad radius*/
792  1               double rcor;/*receiving fiber core radius*/
793  1               double dcle;/*exciting fiber clad diameter*/
794  1               double dclr;/*receiving fiber clad diameter*/
795  1               double thce;/*exciting fiber critical angle*/
796  1               double thcr;/*receiving fiber critical angle*/
797  1               double alpha;/*angle between optical axes*/
798  1               double zcrit;/*value of zr that will force error*/
799  1
800  1               d = fib[4];
801  1               rcle = fib[1]/2;
802  1               rcoe = fib[0]/2;
803  1               rclr = fib[3]/2;
804  1               rcor = fib[2]/2;
805  1               dcle = fib[1];
806  1               dclr = fib[3];
807  1               thce = RADIAN*fib[5];
808  1               thcr = RADIAN*fib[6];
809  1               alpha = RADIAN*fib[7];
810  1               zr = fib[12];
```

```
811     1
812     1               /*test for condition 1*/
813     1               if ((((PI/2) - thce) > alpha) && (zr > 0.0))
814     1                /*condition 1 is possible*/
815     1                {
816     2                 zcrit = (d + rcle - rcoe)*tan((PI/2) - thce);
817     2                 if (zr > zcrit)
818     2                  /*condition 1 has occurred*/
819     2                  {
820     3                   printf("\nError condition 1");
821     3                   printf("\nNear corner of receiving fiber");
822     3                   printf("\nprotrudes into exciting cone\n");
823     3                   locflag = TRUE;
824     3                  }
825     2                }
826     1
827     1               /*test for condition 2*/
828     1               if (((PI/2) - thce) < alpha)
829     1                /*condition 2 is possible*/
830     1                {
831     2                 zcrit = tan((PI/2) - thce);
832     2                 zcrit = zcrit*(dclr*cos(alpha) + d + rcle - rcoe);
833     2                 zcrit = zcrit - dclr*sin(alpha);
834     2                 if (zr > zcrit)
835     2                  /*condition 2 has occurred*/
836     2                  {
837     3                   printf("\nError condition 2");
838     3                   printf("\nFar corner of receiving fiber");
839     3                   printf("\nprotrudes into exciting cone\n");
840     3                   locflag = TRUE;
841     3                  }
842     2                }
843     1
844     1               /*test for condition 3*/
845     1               if (zr < 0.0)
846     1                /*condition 3 is possible*/
847     1                {
848     2                 zcrit = tan((PI/2) - alpha - thcr);
849     2                 zcrit = zcrit*(d + (rclr-rcor)*cos(alpha));
850     2                 zcrit = zcrit + (rclr-rcor)*sin(alpha);
851     2                 if ((-zr) > zcrit)
852     2                  /*condition 3 has occurred*/
853     2                  {
854     3                   printf("\nError condition 3");
855     3                   printf("\nNear corner of exciting fiber");
856     3                   printf("\nprotrudes into receiving cone\n");
857     3                   locflag = TRUE;
858     3                  }
859     2                }
860     1
861     1               /*test for condition 4*/
862     1               if ((alpha + thcr) > (PI/2))
863     1                /*condition 4 is possible*/
864     1                {
865     2                 zcrit = tan(PI - alpha - thcr);
866     2                 zcrit = (dcle + d + (rclr-rcor)*cos(alpha))/zcrit;
867     2                 zcrit = zcrit - (rclr-rcor)*sin(alpha);
```

```
868  2                if (zr < zcrit)
869  2                 /*condition 4 has occurred*/
870  2                 {
871  3                  printf("\nError condition 4");
872  3                  printf("\nFar corner of exciting fiber");
873  3                  printf("\nprotrudes into receiving cone\n");
874  3                  locflag = TRUE;
875  3                 }
876  2               }
877  1
878  1              return(locflag);
879  1       }
880
881             /* function fig_power()                                    */
882             /*                                                         */
883             /* Function that calculates the received Raman power and the*/
884             /* power as a function of the distance along the optical   */
885             /* axis of the exciting fiber.                             */
886             /*                                                         */
887             /* Raman scattering is excited in a truncated right        */
888             /* circular cone shaped space coaxial with the optical axis */
889             /* of the excitation fiber. A given finite subspace of the  */
890             /* cone of excitation is assumed to radiate Raman scattered */
891             /* light with equal intensity in all directions. That is, it*/
892             /* radiates into 4 PI steradians of solid angle.           */
893             /* This function divides the cone of excitation of the     */
894             /* exciting fiber into a finite number of finite elements.  */
895             /* The division is made in cylindrical coordinates with the */
896             /* origin at the center of the face of the exciting fiber.  */
897             /* the optical axis is the z-axis. The theta=0 plane is the */
898             /* plane perpendicular to both the face of the exciting    */
899             /* fiber and the plane containing both the optical axis and */
900             /* the axis of symmetry for the entire system. The space is */
901             /* divided into AXIAL right circular cylinders in the z    */
902             /* direction. The radius of each cylinder is the radius of  */
903             /* the cone at the center of the finite element. Each      */
904             /* cylinder is divided radially into a center cylinder and  */
905             /* RADIAL-1 annuli. In the azimuthal direction, each       */
906             /* cylinder is divided into AZIMUTH sectors.               */
907             /* It also divides the face of the core of the receiving    */
908             /* fiber into a finite number of finite elements.          */
909             /* The receiving fiber face is divided into a circle and    */
910             /* RADIAL-1 annuli. In the azimuthal dimension, the face is */
911             /* divided into AZIMUTH sectors.                           */
912             /* The function finds the power coupled                    */
913             /* from all the elements of the excitation cone to each    */
914             /* element of the receiving face and sums the power received*/
915             /* by all the elements of the receiving face.              */
916             /*                                                         */
917             /* Input data:                                             */
918             /*  fiber[] - properties of probe                          */
919             /*                                                         */
920             /* Output data:                                            */
921             /*  z_pow - Raman power as a function of distance          */
922             /*  coup - received Raman power (power coupled into the    */
923             /*         receiving fiber as a percentage of the power    */
924             /*         fed in by the exciting fiber)                   */
```

```
925           /*                                                          */
926           /* Functions called:                                        */
927           /*  init - Does unit conversion of probe parameters         */
928           /*         Initializes other variables                      */
929           /*                                                          */
930           /*  delta_power - conputes incremental contribution to      */
931           /*                received power in a receiving finite      */
932           /*                element for a given exciting finite       */
933           /*                element.                                  */
934           /*                                                          */
935           /************************************************************/
936
937           fig_power(fiber,z_pow,coup)
938           double fiber[];/*fiber parameters furnished by user*/
939           double z_pow[];/*Raman power as a function of distance*/
940           double *coup;/*received Raman power computed by this function*/
941
942           {
943     1             /* local declarations and initializations */
944     1             double fiber1[34];/*holds all parameters needed by*/
945     1                               /*delta_power. for definitions of*/
946     1                               /*each element, see header block*/
947     1                               /*of init.*/
948     1             int i,iz,ith,ir,irr,ithr;/*loop counters*/
949     1             double big=0.0;/*remembers largest value of power*/
950     1             double rmax;/*maximum radius for a given z*/
951     1             double rmin;/*initial radius*/
952     1
953     1             init(fiber,fiber1);
954     1
955     1               /*inform user of progress*/
956     1             for (i=0;i<34;++i)
957     1             {
958     2                     printf("\n fiber[%d]= %20.12f",i,fiber1[i]);
959     2              }
960     1
961     1             /*step along the z-axis for AXIAL steps*/
962     1             for (iz=1;iz<=AXIAL;++iz)/*step along optical axis*/
963     1              {                        /*of exciting fiber*/
964     2               /*start where cones first intersect*/
965     2               fiber1[28] = fiber1[13] + fiber1[15]*(0.5 + (double)(iz-1));
966     2
967     2               /*calculate maximum radial distance for this z coordinate*/
968     2               rmax = fiber1[0] + fiber1[28]*tan(fiber1[5]);
969     2
970     2               fiber1[31] = rmax/RADIAL;/*set radius increment*/
971     2               rmin = fiber1[31]/2;
972     2               z_pow[(iz-1)] = 0.0;
973     2
974     2               for (ir=1;ir<=RADIAL;++ir) /*step thru radii in exc cone*/
975     2                {
976     3                 fiber1[26] = rmin + (ir-1)*fiber1[31];/*current radius*/
977     3                 for (ith=1;ith<=AZIMUTH;++ith)/*exciting cone angles*/
978     3                  {
979     4                   fiber1[27] = fiber1[16] + (ith-1)*(2*PI/AZIMUTH);
980     4                   for (irr=1;irr<=RADIAL;++irr)/*receiving face radii*/
981     4                    {
```

```
 982     5                    fiber1[29] = fiber1[17] + (irr-1)*(fiber1[2]/RADIAL);
 983     5                    /*step thru receiving face azimuth*/
 984     5                    for (ithr=1;ithr<=AZIMUTH;++ithr)
 985     5                     {
 986     6
 987     6                      /*due to symmetry, only step thru PI radians*/
 988     6
 989     6                      fiber1[30]=fiber1[18]+(ithr-1)*(PI/AZIMUTH);
 990     6
 991     6                      /*The purpose of all the preceding was to find*/
 992     6                      /*the coordinates of the centers of the finite*/
 993     6                      /*elements presently under consideration. Now */
 994     6                      /*that we have them, we can determine the power*/
 995     6                      /*contributed by this particular element of the*/
 996     6                      /*excitation cone to this particular element of*/
 997     6                      /*the receiving face.*/
 998     6
 999     6                      delta_power(fiber1);
1000     6
1001     6                      /*Note: The following summing operation is the*/
1002     6                      /*actual integration. The integral is approximated*/
1003     6                      /*by summing the discrete individual contributions*/
1004     6                      /*to the total power. */
1005     6
1006     6                      fiber1[12] = fiber1[12] + 2*fiber1[33];
1007     6
1008     6                      z_pow[(iz-1)]=z_pow[(iz-1)]+2*fiber1[33];
1009     6                     }
1010     5                   }
1011     4                  }
1012     3                 }
1013     2                 if (z_pow[(iz-1)]>big)
1014     2                       big=z_pow[(iz-1)];
1015     2                       printf("\nz= %f microns, Power= %f\n",fiber1[28]*1000000,fiber1[12]);
1016     2                }
1017     1
1018     1               /*compute received Raman power of probe*/
1019     1               *coup = fiber1[12]/fiber1[32];
1020     1
1021     1       }
1022
1023             /* function results()                                          */
1024             /*                                                             */
1025             /* function to generate output data file                       */
1026             /*                                                             */
1027             /* This function writes the fiber parameters, the computed     */
1028             /* received Raman power, and the Raman power as a function     */
1029             /* of distance to an output file.                              */
1030             /*                                                             */
1031             /* Input data:                                                 */
1032             /*                                                             */
1033             /* Fiber[] - properties of probe                               */
1034             /* z_pow[] - Raman power as a function of distance             */
1035             /* coup - received Raman power                                 */
1036             /*                                                             */
1037             /* Output data (written to file):                              */
1038             /*                                                             */
```

```
1039          /* Fiber[] - properties of probe                              */
1040          /* z_pow[] - Raman power as a function of distance            */
1041          /* coup - received Raman power                                */
1042          /*                                                            */
1043          /* Functions called: None                                     */
1044          /************************************************************/
1045
1046          results(fiber,z_pow,coup,filep)
1047          double fiber[];/*fiber parameters furnished by user*/
1048          double z_pow[];/*Raman power as a function of distance*/
1049          double *coup;/*received Raman power*/
1050          FILE *filep;/*file pointer*/
1051
1052          {
1053    1             /* local declarations and initializations */
1054    1             int i; /*loop counter*/
1055    1             static char prompt[NUMPARMS][STRLEN]/*prompts to user*/
1056    1                  ={
1057    1           "Core diameter of excitation fiber (microns) ",
1058    1           "Cladding dia of excitation fiber (microns)  ",
1059    1           "Core diameter of receiving fiber (microns)  ",
1060    1           "Cladding dia of receiving fiber (microns)   ",
1061    1           "Spacing between fiber ends (microns)        ",
1062    1           "Ext critical angle of excitation fiber (deg)",
1063    1           "Ext critical angle of receiving fiber (deg) ",
1064    1           "Angle between fiber optical axes (degrees)  ",
1065    1           "Raman scattering cross sect (cm^2/molecule sr)",
1066    1           "Number density of scatterers / cm^3         ",
1067    1           "Space constant for attenuation (microns)    ",
1068    1          "Analysis depth along exciting fiber axis (exc core diameters)",
1069    1          "Axial offset of receiving fiber (microns)"
1070    1                    };
1071    1             char uprompt[STRLEN];/*string to print*/
1072    1
1073    1             printf("\nreceived Raman power: %f\n",*coup);
1074    1             /*write fiber parameters to file*/
1075    1             for (i=0;i<NUMPARMS;++i)
1076    1              {
1077    2              strcpy(uprompt,(&prompt[0][0]+STRLEN*i));
1078    2              fprintf(filep,"\n%s %15.3f",uprompt,fiber[i]);
1079    2              }
1080    1             fprintf(filep,"\nreceived Raman power: %f\n",*coup);
1081    1             fprintf(filep,"\n\n");
1082    1             /* write z power array*/
1083    1             fprintf(filep,"\nPower as a function of distance: \n");
1084    1             fprintf(filep,"\nThe excitation cone is approximated");
1085    1             fprintf(filep,"\nby a stack of cylinders. The following");
1086    1             fprintf(filep,"\nlist is the contributions to the total");
1087    1             fprintf(filep,"\nreceived power from each cylinder");
1088    1             fprintf(filep,"\nstarting from the cylinder nearest");
1089    1             fprintf(filep,"\nthe face of the exciting fiber\n\n");
1090    1             for (i=0;i<AXIAL;++i)
1091    1              {
1092    2               fprintf(filep,"%f  ",z_pow[i]);
1093    2               if (((i+1)%5) == 0)
1094    2                    fprintf(filep,"\n");
1095    2              }
```

```
1096    1               fprintf(filep,"\n");
1097    1       }
1098
1099            /* function get_parms()                                         */
1100            /*                                                              */
1101            /* Function to interactively get all parameters needed for      */
1102            /* probe analysis.                                              */
1103            /*                                                              */
1104            /* Function querys user for each item in turn. It returns       */
1105            /* the resulting array to the calling function.                 */
1106            /*                                                              */
1107            /* Input data (furnished by user):          */
1108            /* Each parameter is an element of fibparms[]                     */
1109            /*                                                              */
1110            /* Output data (passed back to calling function):               */
1111            /*  Probe parameters are elements of fibparms[], specifically: */
1112            /*    fibparms[0] - core diameter of excitation fiber (microns) */
1113            /*    fibparms[1] - cladding dia of excitation fiber (microns)  */
1114            /*    fibparms[2] - core diameter of receiving fiber (microns)  */
1115            /*    fibparms[3] - cladding dia of receiving fiber (microns)   */
1116            /*    fibparms[4] - spacing between fiber ends (microns)        */
1117            /*    fibparms[5] - ext critical angle of excitation fiber (deg)*/
1118            /*    fibparms[6] - ext critical angle of receiving fiber (deg) */
1119            /*    fibparms[7] - angle between fiber optical axes (degrees)  */
1120            /*    fibparms[8] - Raman scattering cross sect (cm^2/molecule sr)*/
1121            /*    fibparms[9] - number density of scatterers / cm^3         */
1122            /*    fibparms[10] - space constant for attenuation (microns)   */
1123            /*    fibparms[11] - Analysis depth along exciting fiber axis   */
1124            /*                  (excitation fiber core diameters)         */
1125            /*    fibparms[12] - Receiving fiber axial offset (microns)     */
1126            /*                                                              */
1127            /* Functions called: None                                       */
1128            /*                                                              */
1129            /*--------------------------------------------------------------*/
1130
1131            get_parms(fibparms)
1132            double fibparms[];/*fiber parameters furnished by user*/
1133            {
1134    1               /* local declarations and initializations */
1135    1               int i;/*loop counter*/
1136    1               char uprompt[STRLEN];/*prompt to be displayed*/
1137    1               char seq[81];/*input string from keyboard*/
1138    1               char s1[80];/*dumping ground*/
1139    1               int resp;/* number of successful conversions*/
1140    1               double num;/*input string converted to number*/
1141    1               static char prompt[NUMPARMS][STRLEN]/*prompts to user*/
1142    1                       ={
1143    1        "Core diameter of excitation fiber (microns) ",
1144    1        "Cladding dia of excitation fiber (microns)  ",
1145    1        "Core diameter of receiving fiber (microns)  ",
1146    1        "Cladding dia of receiving fiber (microns)   ",
1147    1        "Spacing between fiber ends (microns)        ",
1148    1        "Ext critical angle of excitation fiber (deg)",
1149    1        "Ext critical angle of receiving fiber (deg) ",
1150    1        "Angle between fiber optical axes (degrees)  ",
1151    1        "Raman scattering cross sect (cm^2/molecule sr)",
1152    1        "Number density of scatterers / cm^3         ",
```

```
1153    1       "Space constant for attenuation (microns)   ",
1154    1      "Analysis depth along exciting fiber axis (exc core diameters)",
1155    1      "Axial offset of receiving fiber (microns)"
1156    1                    );
1157    1
1158    1              /*on each of NUMPARMS passes thru loop prompt*/
1159    1              /*user for and obtain the  value*/
1160    1              /*for a probe parameter*/
1161    1
1162    1              for (i=0;i<NUMPARMS;++i)
1163    1               {
1164    2                      /*copy prompt message from list*/
1165    2                      strcpy(uprompt,(&prompt[0][0]+STRLEN*i));
1166    2                      resp = -1;
1167    2                      /*repeat until a  number is entered*/
1168    2                      while (resp <= 0)
1169    2                       {
1170    3                              printf("\n%s",uprompt);/*print prompt*/
1171    3                              scanf("%s",seq);/*get response as string*/
1172    3                              resp=sscanf(seq,"%f%s",&num,s1);
1173    3                              /*convert string to double if possible*/
1174    3                       }
1175    2                      fibparms[i]=num;/*insert data into array*/
1176    2               }
1177    1              printf("\n");
1178    1       }
1179
1180            /* function get_one()                                          */
1181            /*                                                             */
1182            /* Function to interactively update parameters needed for      */
1183            /* probe analysis.                                             */
1184            /*                                                             */
1185            /* Function querys user for item to update.                   */
1186            /* Then it querys for the update value.                       */
1187            /* It returns the resulting array the calling function.        */
1188            /*                                                             */
1189            /* Input data (furnished by user):          */
1190            /* Each parameter is an element of fibparms[]                     */
1191            /*                                                             */
1192            /* Output data (passed back to calling function):              */
1193            /*  Probe parameters are elements of fibparms[], specifically: */
1194            /*    fibparms[0] - core diameter of excitation fiber (microns) */
1195            /*    fibparms[1] - cladding dia of excitation fiber (microns)  */
1196            /*    fibparms[2] - core diameter of receiving fiber (microns)  */
1197            /*    fibparms[3] - cladding dia of receiving fiber (microns)   */
1198            /*    fibparms[4] - spacing between fiber ends (microns)        */
1199            /*    fibparms[5] - ext critical angle of excitation fiber (deg)*/
1200            /*    fibparms[6] - ext critical angle of receiving fiber (deg) */
1201            /*    fibparms[7] - angle between fiber optical axes (degrees)  */
1202            /*    fibparms[8] - Raman scattering cross sect (cm^2/molecule sr)*/
1203            /*    fibparms[9] - number density of scatterers / cm^3         */
1204            /*    fibparms[10] - space constant for attenuation (microns)   */
1205            /*    fibparms[11] - Analysis depth along exciting fiber axis   */
1206            /*                  (excitation fiber core diameters)          */
1207            /*    fibparms[12] - Receiving fiber axial offset (microns)     */
1208            /*                                                              */
1209            /* Functions called: None                                       */
```

```
1210          /*                                                          */
1211          /************************************************************/
1212
1213          get_one(fibparms)
1214          double fibparms[];/*fiber parameters updated by user*/
1215          {
1216    1             /* local declarations and initializations */
1217    1             int i;/*selection offset counter*/
1218    1             int flag = TRUE;/*TRUE => keep query loop going*/
1219    1             int j;/*loop counter*/
1220    1             char k;/*display value*/
1221    1             char choice;/*menu selection*/
1222    1             char uprompt[STRLEN];/*prompt to be displayed*/
1223    1             char seq[81];/*input string from keyboard*/
1224    1             char s1[80];/*dumping ground*/
1225    1             int resp;/* number of successful conversions*/
1226    1             double num;/*input string converted to number*/
1227    1             static char prompt[NUMPARMS][STRLEN]/*prompts to user*/
1228    1                  ={
1229    1           "Core diameter of excitation fiber (microns) ",
1230    1           "Cladding dia of excitation fiber (microns)  ",
1231    1           "Core diameter of receiving fiber (microns)  ",
1232    1           "Cladding dia of receiving fiber (microns)   ",
1233    1           "Spacing between fiber ends (microns)        ",
1234    1           "Ext critical angle of excitation fiber (deg)",
1235    1           "Ext critical angle of receiving fiber (deg) ",
1236    1           "Angle between fiber optical axes (degrees)  ",
1237    1           "Raman scattering cross sect (cm^2/molecule sr)",
1238    1           "Number density of scatterers / cm^3         ",
1239    1           "Space constant for attenuation (microns)   ",
1240    1          "Analysis depth along exciting fiber axis (exc core diameters)",
1241    1          "Axial offset of receiving fiber (microns)"
1242    1                        );
1243    1
1244    1             while(flag)
1245    1              {
1246    2               /*display menu*/
1247    2               for (j=0;j<NUMPARMS;++j)
1248    2                {
1249    3                 strcpy(uprompt,(&prompt[0][0]+STRLEN*j));
1250    3                 k = 'A' + j;
1251    3                 printf ("\n%c. %s",k,uprompt);
1252    3                }
1253    2               printf("\n");
1254    2               printf("Select from menu (A-M) or (Q)uit this menu ");
1255    2               scanf("%s",seq);
1256    2               sscanf(seq,"%c%s",&choice,s1);
1257    2               choice = toupper(choice);
1258    2               if (choice == 'Q')
1259    2                    flag = FALSE;
1260    2               else
1261    2                {
1262    3                 if ((choice <= 'M') && (choice >= 'A'))
1263    3                  {
1264    4                     i = choice - 'A';
1265    4                     /*copy prompt message from list*/
1266    4                     strcpy(uprompt,(&prompt[0][0]+STRLEN*i));
```

```
1267     4                    resp = -1;
1268     4                    /*repeat until a  number is entered*/
1269     4                    while (resp <= 0)
1270     4                     {
1271     5                            printf("\n%s",uprompt);/*print prompt*/
1272     5                            scanf("%s",seq);/*get response as string*/
1273     5                            resp=sscanf(seq,"%f%s",&num,s1);
1274     5                            /*convert string to double if possible*/
1275     5                     }
1276     4                    fibparms[i]=num;/*insert data into array*/
1277     4                  }
1278     3                 }
1279     2                printf("\n\n");
1280     2               }
1281     1       }
1282             /* function init()                                          */
1283             /*                                                          */
1284             /* Function to generate various initial values needed by    */
1285             /* fig_power function. This function takes predefined       */
1286             /* constants and user furnished parameters, and calculates  */
1287             /* values needed by fig_power function.                     */
1288             /*                                                          */
1289             /* Input data (parameter):                                  */
1290             /*  fibber[] - array of data previously specified by user    */
1291             /*                                                          */
1292             /* Output data (parameter):                                 */
1293             /*  fib1[] - array of data required by fig_power           */
1294             /*           consisting of:                                 */
1295             /*  fib1[0] - exc fiber core radius m (rcoe)               */
1296             /*  fib1[1] - exc fiber clad radius m (rcle)               */
1297             /*  fib1[2] - rec fiber core radius m (rcor and rrmax)     */
1298             /*  fib1[3] - rec fiber clad radius m (rclr)               */
1299             /*  fib1[4] - edge to edge spacing m (d)                   */
1300             /*  fib1[5] - exc fiber ext crit angle rad (thce)          */
1301             /*  fib1[6] - rec fiber ext crit angle rad (thcr)          */
1302             /*  fib1[7] - angle between fiber optical axes rad (alpha)*/
1303             /*  fib1[8] - Raman cross section m^2/molecule sr (beta)   */
1304             /*  fib1[9] - Number density scatterers/m^3 (n)            */
1305             /*  fib1[10]- Space constant of attenuation m (z0)         */
1306             /*  fib1[11]- Analysis depth along exciting fiber axis in */
1307             /*              exciting fiber core diameters (ndia)        */
1308             /*  fib1[12]- Total power received by receiver (power)     */
1309             /*  fib1[13]- Minimum z where cones intersect m (zmin)     */
1310             /*  fib1[14]- Maximum z m (zmax)                           */
1311             /*  fib1[15]- z increment finite element length m (dz)     */
1312             /*  fib1[16]- start value of azimuth angle radians (thmin)*/
1313             /*  fib1[17]- start value of receive radius m  (rrmin and drr/2)    */
1314             /*  fib1[18]- start value of receiving azimuth             */
1315             /*              angle radians                 (thrmin)     */
1316             /*  fib1[19]- end value for receiving azimuth              */
1317             /*              angle radians                 (thrmax)     */
1318             /*  fib1[20]- cosine of external receiving                 */
1319             /*              critical angle              (costhcr)      */
1320             /*   fib1[21]- receiving  fiber offset in x dimension m (x12)*/
1321             /*   fib1[22]- receiving  fiber offset in y dimension m (y12)*/
1322             /*   fib1[23]- receiving  fiber offset in z dimension m (z12)*/
1323             /*  fib1[24]- z offset of edge of receiving fiber with     */
```

```
1324            /*            respect to exciting fiber m (zr)            */
1325            /*  fib1[25]-reserved for future use  (nrx)       */
1326            /*fib1[26]-polar radius from optical axis of exciting fiber*/
1327            /*         to center of current exciting finite element m (r) */
1328            /*fib1[27]-polar azimuth from theta=0 plane of exciting fiber*/
1329            /*         to center of current exciting finite element rad (th)*/
1330            /*fib1[28]-axial distance from face of exciting fiber to center*/
1331            /*          of current exciting finite element m              (z)*/
1332            /*fib1[29]-polar radius from optical axis of receiving fiber to*/
1333            /*          center of current receiving finite element m (rr)*/
1334            /*fib1[30]-polar azimuth from theta=0 plane of receiving fiber*/
1335            /*        to center of current receiving finite element rad (thr)*/
1336            /*fib1[31]-incremental polar radius of current exciting finite*/
1337            /*          element m                                      (dr)*/
1338            /*fib1[32]-light power in exciting fiber watts (power0) */
1339            /*fib1[33]-Raman power collected by current receiving finite */
1340            /*          element generated by current exciting finite element*/
1341            /*          watts                                        (delpower)*/
1342            /*                                                              */
1343            /* Functions called: None                                       */
1344            /*                                                              */
1345            /*--------------------------------------------------------------*/
1346
1347            init(fibber,fib1)
1348            double fibber[];/*fiber parameters furnished by user*/
1349            double fib1[];/*fiber parameters needed by delta_power*/
1350
1351            {
1352    1               /*local declarations and initializations*/
1353    1               int i;/*loop counter*/
1354    1               static double factor[NUMPARMS]
1355    1                       = { 0.0000005,
1356    1                           0.0000005,
1357    1                           0.0000005,
1358    1                           0.0000005,
1359    1                           0.000001,
1360    1                           0.01745329251994329591,
1361    1                           0.01745329251994329591,
1362    1                           0.01745329251994329591,
1363    1                           0.0001,
1364    1                           1000000.000000000,
1365    1                           0.000001,
1366    1                           1.0 ); /*unit conversion factors*/
1367    1               double m1,m2,b1,b2,xmin;/*intermediate values for zmin calc*/
1368    1               double f1,t1,denom,t2,numer,zm;/*intermediate values used*/
1369    1                                              /*in zmin zmax calculations*/
1370    1               double dth;/*incremental angle of exciting finite element*/
1371    1               double drr;/*incremental radius of receiving finite element*/
1372    1               double dthr;/*incremental angle of receiving finite element*/
1373    1
1374    1               /*unit conversion of user furnished parameters */
1375    1               for (i=0;i<(NUMPARMS-1);++i)
1376    1                       fib1[i] = fibber[i]*factor[i];
1377    1
1378    1               /*zero out parameters that will be computed by other functions*/
1379    1               for (i=26;i<34;++i)
1380    1                       fib1[i]=0.0;
```

```
1381    1
1382    1               /*initialize other parameters*/
1383    1               fib1[12] = 0.0; /*power=0*/
1384    1               fib1[32]=1.0;/*power0=1 for normalization*/
1385    1               fib1[24] = 0.000001*fibber[12];/*zr*/
1386    1               fib1[25] = 0.0;/*nrx*/
1387    1
1388    1               /*compute zmin*/
1389    1               if ((fib1[7]+fib1[6]) > (PI/2))
1390    1                {
1391    2                 /*this is the rare but possible occurrence in which*/
1392    2                 /*the minimum value of z is determined by the */
1393    2                 /*intersection of the innermost ray of the receiving*/
1394    2                 /*fiber and the outermost ray of the exciting fiber*/
1395    2                 t1=tan(fib1[6]+fib1[7]-(PI/2.0));
1396    2                 f1=((fib1[3]-fib1[2])*cos(fib1[7])+fib1[4]+fib1[1]+fib1[0])*t1;
1397    2                 numer = (fib1[3]-fib1[2])*sin(fib1[7])+fib1[24]-f1;
1398    2                 denom = 1 + tan(fib1[5])*t1;
1399    2                 fib1[13] = numer/denom;
1400    2                }
1401    1               else
1402    1                {
1403    2                       /*this is the more common case*/
1404    2                       /*zmin is determined by the intersection*/
1405    2                       /*of the innermost rays of both fibers*/
1406    2                       m1 = tan(fib1[7]+ fib1[6]);
1407    2                       m2 = m1  + tan(fib1[5]);/*denominator*/
1408    2                       b1 = m1*((fib1[3] - fib1[2])*sin(fib1[7]) + fib1[24]);
1409    2                       b2 = (fib1[3] - fib1[2])*cos(fib1[7]);
1410    2                       b2 = b2 + fib1[4] + fib1[1] - fib1[0];
1411    2                       xmin = b1 + b2;/*numerator*/
1412    2                       fib1[13] = xmin/m2;/* computation of zmin*/
1413    2                }
1414    1
1415    1               /*compute zmax*/
1416    1               fib1[14] = fib1[13]+fib1[11]*2*fib1[0];/*zmax*/
1417    1               if (fib1[7] > (fib1[6]+fib1[5]))
1418    1                {
1419    2                 /*in this case z has a geometrically limited maximum*/
1420    2                 t1 = tan((PI/2.0) - fib1[7] + fib1[6]);
1421    2                 denom = 1 - tan(fib1[5])*t1;
1422    2                 t2 = t1*((fib1[3]+fib1[2])*cos(fib1[7])+fib1[4]+fib1[1]+fib1[0]);
1423    2                 numer = fib1[24]+(fib1[3]+fib1[2])*sin(fib1[7])+t2;
1424    2                 zm = numer/denom;
1425    2                 if (fib1[14] > zm)
1426    2                       fib1[14] = zm;
1427    2                 /*the idea here is that if the number of core diameters*/
1428    2                 /*specified by the user leads to a zmax less than the*/
1429    2                 /*geometrically limited value, the program uses the*/
1430    2                 /*specified zmax instead of the larger geometric zmax*/
1431    2                 /*the reason for doing this is that if the user is */
1432    2                 /*studying the performance of a probe over a range of*/
1433    2                 /*angles and suddenly passes the threshold for which*/
1434    2                 /*there is a natural limit to zmax, the caluculations*/
1435    2                 /*will still have about the same degree of accuracy*/
1436    2                }
1437    1
```

```
1438  1              /*other parameters*/
1439  1              fib1[15] = (fib1[14] - fib1[13])/AXIAL;/*dz*/
1440  1              dth = 2*PI/AZIMUTH;
1441  1              fib1[16]= dth/2;
1442  1              drr = fib1[2]/RADIAL;
1443  1              fib1[17] = drr/2;/*rrmin*/
1444  1              dthr = PI/AZIMUTH;
1445  1              fib1[18] = -(PI/2)+dthr/2;/*thrmin*/
1446  1              fib1[19] = PI/2;/*thrmax*/
1447  1              fib1[20] = cos(fib1[6]);/*costhcr*/
1448  1              fib1[21] = 0.0;/*x12*/
1449  1              fib1[22] = fib1[1]+fib1[4]+fib1[3]*cos(fib1[7]);/*y12*/
1450  1              fib1[23] = fib1[3]*sin(fib1[7]);/*z12*/
1451  1
1452  1      )
1453
1454          /* function delta_power()                                    */
1455          /* Function determines the contribution of power from the    */
1456          /* current exciting finite element to the current receiving  */
1457          /* finite element.                                           */
1458          /*                                                           */
1459          /* Function figures the Raman scattered power per unit volume*/
1460          /* at the center of the current exciting finite element. Then*/
1461          /* it figures the volume. The product of these is the total  */
1462          /* power radiated into 4 PI steradians of space by the current*/
1463          /* finite element. Fuinction then figures the amount of solid */
1464          /* angle intercepted by the current receiving finite element. */
1465          /* From this it figures the fraction of power radiated from  */
1466          /* the exciting finite element that is intercepted by the    */
1467          /* receiving finite element.                                 */
1468          /*                                                           */
1469          /* Input data (parameter):                                   */
1470          /*  fib1[] - properties of probe and coordinates of centers  */
1471          /*           of finite elements                              */
1472          /*           see header block of init() for definitions      */
1473          /*           of individual array elements                    */
1474          /*                                                           */
1475          /* Output data (parameter):                                  */
1476          /*  fib1[33] - power intercepted by receiving finite element */
1477          /*                                                           */
1478          /* Functions called: None                                    */
1479          /*************************************************************/
1480
1481
1482          delta_power(fib1)
1483          double fib1[];/*fiber parameters*/
1484
1485          {
1486  1               /*local declarations and initializations*/
1487  1               int i;/*loop counter*/
1488  1               double x,y,z;/*rectangular coordinates with respect to */
1489  1                     /*the center of the exciting face*/
1490  1               double xr,yr;/*rectangular coordinates with respect to */
1491  1                     /*the center of the receiving face*/
1492  1               double drrx;/*distance between finite elements in x direction*/
1493  1               double drry;/*distance between finite elements in y direction*/
1494  1               double drrz;/*distance between finite elements in z direction*/
```

```
1495  1             double magdrr;/*distance between centers of finite elements*/
1496  1             double nrrz;/*z component of unit vector on receiving face*/
1497  1             double nrry;/*y component of unit vector on receiving face*/
1498  1             double nrrx=0.0;/*x component of unit vector on receiving face*/
1499  1             double costhrr;/*cosine of angle between line connecting  */
1500  1             /*finite  elements and line normal to receiving face*/
1501  1             double attnz;/*attenuation factor for this region of space*/
1502  1             double abig;/*area enclosed by circle that bounds finite */
1503  1                     /*element from the outside*/
1504  1             double asmall;/*area enclosed by circle that bounds finite */
1505  1                     /*element from the inside*/
1506  1             double dvol;/*volume of exciting finite element*/
1507  1             double dth;/*increment of azimuth*/
1508  1             double delirrad;/*power radiated into 4 PI steradians*/
1509  1             double dthr;/*incemental angle of receiving element*/
1510  1             double dar;/*incremental area of receiving element*/
1511  1             double delsr;/*fraction of power intercepted by */
1512  1                     /*receiving finite element*/
1513  1
1514  1             /* geometric calculations */
1515  1             x = fib1[26]*cos(fib1[27]);
1516  1             y = fib1[26]*sin(fib1[27]);
1517  1             z = fib1[28];
1518  1             xr = fib1[29]*cos(fib1[30]);
1519  1             yr = fib1[29]*sin(fib1[30]);
1520  1             drrx = x - (xr + fib1[21]);
1521  1             drry = y - (yr*cos(fib1[7])+fib1[22]);
1522  1             drrz = z - (fib1[24]+fib1[23]+yr*sin(fib1[7]));
1523  1             magdrr = drrx*drrx;
1524  1             magdrr = magdrr + drry*drry;
1525  1             magdrr = magdrr + drrz*drrz;
1526  1             magdrr = sqrt(magdrr);
1527  1             nrrz = cos(fib1[7]);
1528  1             nrry = -sin(fib1[7]);
1529  1             costhrr = (drrx*nrrx+drry*nrry+drrz*nrrz);
1530  1             costhrr = costhrr/magdrr;
1531  1     /*
1532  1             printf("\ncosthrr = %16.12f magdrr = %16.12f",costhrr,magdrr);
1533  1     */
1534  1
1535  1             /*is this exciting element within the cone*/
1536  1             /*of acceptance of this receiving element?*/
1537  1
1538  1             if (costhrr > fib1[20])
1539  1              {
1540  2                /*exciting element is in cone of acceptance*/
1541  2                /*figure power collected*/
1542  2                /*attenuation occurs along the entire path*/
1543  2                attnz = exp(-(z+magdrr)/fib1[10]);
1544  2                abig=PI*((fib1[26]+(fib1[31]/2))*(fib1[26]+(fib1[31]/2)));
1545  2                asmall=PI*((fib1[26]-(fib1[31]/2))*(fib1[26]-(fib1[31]/2)));
1546  2                dth = 2*PI/AZIMUTH;
1547  2                dvol = fib1[15]*(abig-asmall)*dth/(2*PI);
1548  2     /*
1549  2     printf("\n\n\n\n\n");
1550  2             printf("\nabig= %16.12f asmall= %16.12f",abig,asmall);
1551  2     */
```

```
1552    2               delirrad=fib1[32]*fib1[8]*fib1[9]*dvol*attnz;
1553    2               delirrad=delirrad/(PI*fib1[31]*RADIAL*fib1[31]*RADIAL);
1554    2               abig=PI*((fib1[29]+(fib1[17]))*(fib1[29]+(fib1[17])));
1555    2               asmall=PI*((fib1[29]-(fib1[17]))*(fib1[29]-(fib1[17])));
1556    2               dthr=PI/AZIMUTH;
1557    2               dar=(abig-asmall)*dthr/(2*PI);
1558    2               delsr=dar*costhrr/(magdrr*magdrr);
1559    2               fib1[33]=delsr*delirrad;
1560    2       /*
1561    2               printf("\nabig= %16.12f asmall= %16.12f",abig,asmall);
1562    2               printf("\ndvol= %16.12f dar= %16.12f",dvol,dar);
1563    2               printf("\ndelirrad = %16.12f delsr = %16.12f",delirrad,delsr);
1564    2       */
1565    2             }
1566    1           else
1567    1             {
1568    2             /*exciting element outside cone of acceptance*/
1569    2             /*no power contributed by this exciting element*/
1570    2             fib1[33] = 0.0;
1571    2       /*
1572    2       printf("\nExciting element outside receiving cone");
1573    2       */
1574    2             }
1575    1              /*inform user of progress*/
1576    1       /*
1577    1           for (i=0;i<34;++i)
1578    1           {
1579    1                   printf("\n fiber[%d]= %16.12f",i,fib1[i]);
1580    1             }
1581    1       */
1582    1
1583    1       }
```

Command Line

```
CC PROBE/LIST
```

```
University of Tennessee VAXcluster member UTKVX3

Username: SKERCEL
Password:
        Welcome to VAX/VMS version V5.2 on node UTKVX3
     Last interactive login on Thursday,  8-FEB-1990 13:27
         Group 14143 User 133208 logged in Thursday,  8-FEB-1990 13:30
1m
[08-Feb-90] DCA Network Disconnect Character to change on Feb. 12, 1990
[07-Jan-90] JNET Version 3.4 installed on VAXcluster
[20-Dec-89] KERMIT default parity set to SPACE
[14-Dec-89] UTCC Spring Short Courses Schedule online. Type HELP SHORT_COURSES.
[22-Nov-89] I-DEAS Plotting Capabilities Enhanced
[21-Nov-89] GCG Sequence Analysis Software Package available on VAXcluster
[16-Nov-89] ARC/INFO Version 5.0 installed
[19-Nov-89] UTCC Changed Its Network Parity on Sunday November 19th
[15-Nov-89] Notice of VAX2 and VAX4 Unavailability, Mon.-Sat.,6-6:30am
[02-Nov-89] PRECAL supports color for Tektronix 41xx and 42xx terminals
[02-Nov-89] ACSL upgraded from Level 8R to Level 9B
[30-Oct-89] VAXcluster now open to users 4am-8am Monday-Saturday
[29-Oct-89] DCA destination VAX selects either VAX2 or VAX4

%NOTICE-I-MORE, Type "NOTICE" for full display of notices
0m

62"p F

Where do you want your printouts? Enter remote number: 1
$ set default tmp$work1:[space1.skercel]
$ dir

Directory TMP$WORK1:[SPACE1.SKERCEL]

PROBE.C;1              93/93       8-FEB-1990 13:25:34.52  [SKERCEL]   (RWED,RWED,RE,)
RAY.FOR;1              36/36       8-FEB-1990 13:26:00.76  [SKERCEL]   (RWED,RWED,RE,)
SETHOST.LOG;1           0/33       8-FEB-1990 13:30:18.63  [SKERCEL]   (RWED,RWED,RE,)

Total of 3 files, 129/162 blocks.
$ cc probe
$ del probe.obj;1
$ cc probe /list
$ link probe
$ run probe


University of Tennessee
Measurement and Control Engineering Center

Probe analysis program
Algorithm developed by M J Roberts
VAX C implementation by S W Kercel
User supplies probe data interactively


 WARNING AND DISCLAIMER:
 This program is in the public domain, and is offered free
 of charge on an AS IS basis. It has been checked with
 representative data, and it returns results consistent
```

```
 with experimental observations.

 HOWEVER,

 the authors make no guarantees concerning its performance.
 Neither the authors nor The University of Tennessee
 assume any liability for damages that may result from
 the use of this program.


(C)ontinue or (Q)uit program?
c
Core diameter of excitation fiber (microns)
400
Cladding dia of excitation fiber (microns)
500
Core diameter of receiving fiber (microns)
400
Cladding dia of receiving fiber (microns)
500
Spacing between fiber ends (microns)
1000
Ext critical angle of excitation fiber (deg)
10
Ext critical angle of receiving fiber (deg)
11
Angle between fiber optical axes (degrees)
12
Raman scattering cross sect (cm^2/molecule sr)
1
Number density of scatterers / cm^3
1
Space constant for attenuation (microns)
1000000
Analysis depth along exciting fiber axis (exc core diameters)
100
Axial offset of receiving fiber (microns)
0


 fiber[0]=         0.000200000000
 fiber[1]=         0.000250000000
 fiber[2]=         0.000200000000
 fiber[3]=         0.000250000000
 fiber[4]=         0.001000000000
 fiber[5]=         0.174532925199
 fiber[6]=         0.191986217719
 fiber[7]=         0.209439510239
 fiber[8]=         0.000100000000
 fiber[9]= 1000000.000000000000
 fiber[10]=          1.000000000000
 fiber[11]=        100.000000000000
 fiber[12]=          0.000000000000
 fiber[13]=          0.001836412690
 fiber[14]=          0.041836412690
 fiber[15]=          0.000500000000
 fiber[16]=          0.196349540849
 fiber[17]=          0.000025000000
```

```
 fiber[18]=       -1.472621556370
 fiber[19]=        1.570796326795
 fiber[20]=        0.981627183448
 fiber[21]=        0.000000000000
 fiber[22]=        0.001494536900
 fiber[23]=        0.000051977923
 fiber[24]=        0.000000000000
 fiber[25]=        0.000000000000
 fiber[26]=        0.000000000000
 fiber[27]=        0.000000000000
 fiber[28]=        0.000000000000
 fiber[29]=        0.000000000000
 fiber[30]=        0.000000000000
 fiber[31]=        0.000000000000
 fiber[32]=        1.000000000000
 fiber[33]=        0.000000000000
z= 2086.412690 microns, Power= 0.000006

z= 2586.412690 microns, Power= 0.000078

z= 3086.412690 microns, Power= 0.000214

z= 3586.412690 microns, Power= 0.000380

z= 4086.412690 microns, Power= 0.000553

z= 4586.412690 microns, Power= 0.000723

z= 5086.412690 microns, Power= 0.000883

z= 5586.412690 microns, Power= 0.001033

z= 6086.412690 microns, Power= 0.001170

z= 6586.412690 microns, Power= 0.001296

z= 7086.412690 microns, Power= 0.001407

z= 7586.412690 microns, Power= 0.001501

z= 8086.412690 microns, Power= 0.001581

z= 8586.412690 microns, Power= 0.001650

z= 9086.412690 microns, Power= 0.001709
```

z= 9586.412690 microns, Power= 0.001761

z= 10086.412690 microns, Power= 0.001807

z= 10586.412690 microns, Power= 0.001848

z= 11086.412690 microns, Power= 0.001885

z= 11586.412690 microns, Power= 0.001917

z= 12086.412690 microns, Power= 0.001946

z= 12586.412690 microns, Power= 0.001972

z= 13086.412690 microns, Power= 0.001996

z= 13586.412690 microns, Power= 0.002018

z= 14086.412690 microns, Power= 0.002038

z= 14586.412690 microns, Power= 0.002056

z= 15086.412690 microns, Power= 0.002074

z= 15586.412690 microns, Power= 0.002090

z= 16086.412690 microns, Power= 0.002105

z= 16586.412690 microns, Power= 0.002118

z= 17086.412690 microns, Power= 0.002131

z= 17586.412690 microns, Power= 0.002143

z= 18086.412690 microns, Power= 0.002155

z= 18586.412690 microns, Power= 0.002165

z= 19086.412690 microns, Power= 0.002174

z= 19586.412690 microns, Power= 0.002184

z= 20086.412690 microns, Power= 0.002192

z= 20586.412690 microns, Power= 0.002200

z= 21086.412690 microns, Power= 0.002207

z= 21586.412690 microns, Power= 0.002214

z= 22086.412690 microns, Power= 0.002221

z= 22586.412690 microns, Power= 0.002227

z= 23086.412690 microns, Power= 0.002233

z= 23586.412690 microns, Power= 0.002238

z= 24086.412690 microns, Power= 0.002243

z= 24586.412690 microns, Power= 0.002248

z= 25086.412690 microns, Power= 0.002253

z= 25586.412690 microns, Power= 0.002258

z= 26086.412690 microns, Power= 0.002262

z= 26586.412690 microns, Power= 0.002266

z= 27086.412690 microns, Power= 0.002270

z= 27586.412690 microns, Power= 0.002274

z= 28086.412690 microns, Power= 0.002277

z= 28586.412690 microns, Power= 0.002281

z= 29086.412690 microns, Power= 0.002284

z= 29586.412690 microns, Power= 0.002288

z= 30086.412690 microns, Power= 0.002291

z= 30586.412690 microns, Power= 0.002294

z= 31086.412690 microns, Power= 0.002297

z= 31586.412690 microns, Power= 0.002300

z= 32086.412690 microns, Power= 0.002302

z= 32586.412690 microns, Power= 0.002305

z= 33086.412690 microns, Power= 0.002308

z= 33586.412690 microns, Power= 0.002310

z= 34086.412690 microns, Power= 0.002313

z= 34586.412690 microns, Power= 0.002315

z= 35086.412690 microns, Power= 0.002317

z= 35586.412690 microns, Power= 0.002319

z= 36086.412690 microns, Power= 0.002321

z= 36586.412690 microns, Power= 0.002323

z= 37086.412690 microns, Power= 0.002325

z= 37586.412690 microns, Power= 0.002327

z= 38086.412690 microns, Power= 0.002329

z= 38586.412690 microns, Power= 0.002331

z= 39086.412690 microns, Power= 0.002333

```
z= 39586.412690 microns, Power= 0.002334

z= 40086.412690 microns, Power= 0.002336

z= 40586.412690 microns, Power= 0.002338

z= 41086.412690 microns, Power= 0.002339

z= 41586.412690 microns, Power= 0.002341

received Raman power: 0.002341


(C)ontinue or (Q)uit program?
q
$ type probe.dat

Core diameter of excitation fiber (microns)          400.000
Cladding dia of excitation fiber (microns)           500.000
Core diameter of receiving fiber (microns)           400.000
Cladding dia of receiving fiber (microns)            500.000
Spacing between fiber ends (microns)                1000.000
Ext critical angle of excitation fiber (deg)          10.000
Ext critical angle of receiving fiber (deg)           11.000
Angle between fiber optical axes (degrees)            12.000
Raman scattering cross sect (cm^2/molecule sr)            1.000
Number density of scatterers / cm^3                       1.000
Space constant for attenuation (microns)        1000000.000
Analysis depth along exciting fiber axis (exc core diameters)          100.000
Axial offset of receiving fiber (microns)             0.000
received Raman power: 0.002341


Power as a function of distance:

The excitation cone is approximated
by a stack of cylinders. The following
list is the contributions to the total
received power from each cylinder
starting from the cylinder nearest
the face of the exciting fiber

0.000006  0.000072  0.000136  0.000165  0.000174
0.000169  0.000160  0.000149  0.000137  0.000126
0.000111  0.000094  0.000080  0.000068  0.000059
0.000052  0.000046  0.000041  0.000036  0.000032
0.000029  0.000026  0.000024  0.000022  0.000020
0.000019  0.000017  0.000016  0.000015  0.000014
0.000013  0.000012  0.000011  0.000010  0.000010
0.000009  0.000008  0.000008  0.000007  0.000007
0.000007  0.000006  0.000006  0.000006  0.000005
0.000005  0.000005  0.000005  0.000004  0.000004
```

```
0.000004  0.000004  0.000004  0.000004  0.000003
0.000003  0.000003  0.000003  0.000003  0.000003
0.000003  0.000003  0.000003  0.000002  0.000002
0.000002  0.000002  0.000002  0.000002  0.000002
0.000002  0.000002  0.000002  0.000002  0.000002
0.000002  0.000002  0.000002  0.000002  0.000002

$ log
Accounting information:
Buffered I/O count:         1784     Peak WS size:              3108
Direct I/O count:            417     Peak pagefile size:        7856
Page faults:                6033     Mounted volumes:              0
Charged CPU time   0 00:01:36.45  Elapsed time:      0 00:07:15.11

Charge information:
Buffered I/O:      $     0.00000NC   Peak WS:          $     0.00000NC
Direct I/O:              0.04587     Peak pagefile:           0.00000NC
Page faults:             0.00000NC   Mounted volumes:         0.00000
Charged CPU:             2.35552     Elapsed time:            0.04229
Total Charges:     $     2.44368
  SKERCEL       logged out at  8-FEB-1990 13:37:32.70
```

APPENDIX I

CODE LISTING: DOUBLE LENS COUPLER DESIGN PROGRAM

```
0001    c RAY.FOR
0002    c
0003    c program to do repeated ray tracing
0004    c
0005    c S. W. Kercel                          10/24/89
0006    c
0007    c Measurement and Control Engineering Center
0008    c
0009    c
0010    c Warning and Disclaimer:
0011    c
0012    c This program is in the Public Domain and is offered free of
0013    c charge on an AS IS basis. It has been checked with representative
0014    c data, and it returns results consistent with experimental
0015    c observations.
0016    c
0017    c HOWEVER,
0018    c
0019    c The author makes no guarantees concerning its performance.
0020    c Neither the author nor the University of Tennessee assume
0021    c any liability for damages that may result from the use of
0022    c this program.
0023    c
0024    c Description:
0025    c
0026    c This program traces light rays through an optical system
0027    c consisting of two lenses and three open spaces,
0028    c ->|)->(|->.The user
0029    c specifies ranges of focal lengths, lens spacing, and distance
0030    c from second lens to image. For each combination, the program
0031    c will compute the distance from the source to the first lens,
0032    c and the displacement and ray angles of the imaged point.
0033    c It also compares the displacement and angles of the image
0034    c rays with a predetermined criterion. If the properties of
0035    c the image rays are smaller than the criterion, then the
0036    c geometry is saved to GOODRAY.DAT.
0037    c In optical terms, if the instrument is not overfilled,
0038    c the results are stored in GOODRAY.DAT.
0039    c Data for all rays are saved to ALLRAY.DAT.
0040    c
0041    c Input Data:
0042    c
0043    c All input data are specified in the source code and are
0044    c compiled with the program.
0045    c
0046    c d5max - maximum distance (mm) from second lens to image
0047    c d5min - minimum distance (mm) from second lens to image
0048    c d5inc - incremental distance (mm) from 2nd lens to image
0049    c f4max - maximum focal length (mm) of 2nd lens
0050    c f4min - minimum focal length (mm) of 2nd lens
0051    c f4inc - incremental focal length (mm) of 2nd lens
0052    c d3max - maximum spacing (mm) between lenses
0053    c d3min - minimum spacing (mm) between lenses
0054    c d3inc - incremental spacing (mm) between lenses
0055    c f2max - maximum focal length (mm) of 1st lens
0056    c f2min - minimum focal length (mm) of 1st lens
0057    c f2inc - incremental focal length (mm) of 1st lens
```

```
0058     c ri - distance (mm) from source to optical axis
0059     c ri1prime - angle (radians) from 1st source ray to optical axis
0060     c ri2prime - angle (radians) from 2nd source ray to optical axis
0061     c ri3prime - angle (radians) from 3rd source ray to optical axis
0062     c rocrit - maximum allowable distance (mm) from image
0063     c          to optical axis
0064     c roprimecr - maximum allowable angle (radians) from image
0065     c             ray to optical axis
0066     c
0067     c Output Data:
0068     c
0069     c Data for all image rays is saved to ALLRAY.DAT.
0070     c Data for image rays with properties less than or
0071     c equal to the criterion is also saved to GOODRAY.DAT.
0072     c
0073     c rd5 - distance (mm) from second lens to image
0074     c rf4 - focal length (mm) of second lens
0075     c rd3 - spacing (mm) between lenses
0076     c rf2 - focal length (mm) of first lens
0077     c rd1 - distance (mm) from source to first lens
0078     c ro - distance (mm) from image to optical axis
0079     c ro1prime - angle (radians) of 1st image ray with optical axis
0080     c ro2prime - angle (radians) of 2nd image ray with optical axis
0081     c ro3prime - angle (radians) of 3rd image ray with optical axis
0082     c
0083     c Subprograms called from main program:
0084     c
0085     c RAYSEQ - checks properties of source ray data
0086     c COMP_D1 - finds distance from source to first lens and
0087     c           properties of image rays
0088     c RAYSGOOD - checks whether rays meet criterion
0089     c etendoit - checks if current error is lowest encountered
0090     c etendue - calculates light gathering power
0091     c MOVEDATA - copies data from current geometry to best geometry
0092     c
0093     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0094           program ray
0095     c
0096     c declarations
0097     c
0098           implicit none
0099     c loop counters
0100     c
0101           integer i1,i2,i3,i4
0102     c
0103     c user specified data (see header block for descriptions)
0104     c
0105           integer d5max/320/,d5min/290/,d5inc/1/
0106           integer f4max/100/,f4min/100/,f4inc/25/
0107           integer d3max/120/,d3min/100/,d3inc/1/
0108           integer f2max/75/,f2min/75/,f2inc/25/
0109           real ri/0.8/,ri1prime/0.225/,ri2prime/0.0/,ri3prime/-0.225/
0110           real rocrit/2.5/,roprimecr/0.0796/
0111     c
0112     c output data (see header block for descriptions)
0113     c
0114           real rd5,rf4,rd3,rf2,rd1,ro,ro1prime,ro2prime,ro3prime
```

```
0115     c
0116     c intermediate computation data
0117     c
0118           logical rayflag ! .TRUE. => ray angles in proper sequence
0119           logical goodflag ! .TRUE. => output ray meets criterion
0120           logical firstflag/.TRUE./ !.TRUE.=> first pass thru loop
0121           logical bestflag ! .TRUE. => current product smallest so far
0122           real rd1opt ! best spacing from source to first lens
0123           real rf2opt ! best focal length for first lens
0124           real rd3opt ! best spacing betweeen lenses
0125           real rf4opt ! best focal length for second lens
0126           real rd5opt ! best spacing from second lens to image
0127           real roopt ! height of best image
0128           real ro1opt ! first ray angle of best image
0129           real ro2opt ! second ray angle of best image
0130           real ro3opt ! third ray angle of best image
0131           real etendue1 ! etendue of source
0132           real etendue2 ! etendue of target
0133     c
0134     c functions
0135     c
0136           logical rayseq,raygeo,raysgood,etendoit
0137           real etendue
0138     c
0139     c open output data files
0140     c
0141           open (12, file = 'GOODRAY.DAT',status = 'NEW')
0142           open (11, file = 'ALLRAY.DAT',status = 'NEW')
0143     c
0144     c if rays are ok then set up 4 deep nested loop to
0145     c analyze each combination of lens geometry
0146     c
0147           rayflag = rayseq(ri1prime,ri2prime,ri3prime,ri)
0148           if (rayflag) then
0149            write (11,102) ri,ri1prime,ri2prime,ri3prime
0150            write (12,102) ri,ri1prime,ri2prime,ri3prime
0151            etendue1 = etendue(ri,ri1prime)
0152            etendue2 = etendue(rocrit,roprimecr)
0153            write (12,104) etendue1,etendue2
0154            write (11,101)
0155            write (12,101)
0156            do i1 = d5min,d5max,d5inc
0157             do i2 = f4min,f4max,f4inc
0158              do i3 = d3min,d3max,d3inc
0159               do i4 = f2min,f2max,f2inc
0160                rd5 = float(i1)
0161                rf4 = float(i2)
0162                rd3 = float(i3)
0163                rf2 = float(i4)
0164                 call comp_d1(rf2,rd3,rf4,rd5,rd1,ri,ri1prime,
0165          &ri2prime,ri3prime,ro,ro1prime,ro2prime,ro3prime)
0166                 write (11,100) rd1,rf2,rd3,rf4,rd5,ro,ro1prime,
0167          &ro2prime,ro3prime
0168     c
0169     c ray meets criterion => write to goodfile
0170     c
0171                 goodflag = raysgood(ro,rocrit,ro1prime,roprimecr,
```

```
0172          &ro2prime,ro3prime)
0173                  if (goodflag) then
0174                   write (12,100) rd1,rf2,rd3,rf4,rd5,ro,ro1prime,
0175          &ro2prime,ro3prime
0176                  end if ! goodflag
0177      c
0178      c on first pass write data to minimum etendoit variables
0179      c on succeeding passes retain minimum values encountered
0180      c
0181                  if (firstflag) then
0182                   firstflag = .FALSE.
0183                   call movedata(rd1opt,rf2opt,rd3opt,rf4opt
0184          &,rd5opt,roopt,ro1opt,ro2opt,ro3opt,rd1,rf2,rd3,rf4,
0185          &rd5,ro,ro1prime,ro2prime,ro3prime)
0186                  else
0187                   bestflag = etendoit(ro,ro1prime,rocrit,
0188          &roprimecr,roopt,ro1opt)
0189                   if (bestflag) then
0190                    call movedata(rd1opt,rf2opt,rd3opt,rf4opt
0191          &,rd5opt,roopt,ro1opt,ro2opt,ro3opt,rd1,rf2,rd3,rf4,
0192          &rd5,ro,ro1prime,ro2prime,ro3prime)
0193                   end if ! bestflag
0194                  end if !firstflag
0195                end do
0196               end do
0197              end do
0198             end do
0199             write (12,103)
0200             write (12,101)
0201             write (12,100) rd1opt,rf2opt,rd3opt,rf4opt,rd5opt,
0202          &roopt,ro1opt,ro2opt,ro3opt
0203           else
0204             print *,'Ray angles must be in sequence'
0205             print *,'RI1PRIME > RI2PRIME > RI3PRIME'
0206             print *,'RI > 0'
0207           end if ! rayflag
0208      c
0209      c final cleanup
0210      c
0211      100    format (1X,9(F10.3,X))
0212      101    format (1X,'         D1          F2          D3
0213           &F4          D5       Height      Angle 1     Angle 2
0214           &    Angle 3')
0215      102    format (1X,'Input Ray Height: ',F6.3,1X,'Ray Angles: ',
0216           &3(F7.4,1X))
0217      103    format (1X,'Minimum Error of Angle 1 and Height:')
0218      104    format (1X,'Etendue of Source: 'F6.3,1X,
0219           &'Etendue of Target: ',f6.3)
0220           close (11)
0221           close (12)
0222           end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	950	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	353	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	548	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	1851	

ENTRY POINTS

Address	Type	Name
0-00000000		RAY

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name	Address	Type	Name
**	L*4	BESTFLAG	**	I*4	D3INC	**	I*4	D3MAX	**	I*4	D3MIN
**	I*4	D5INC	**	I*4	D5MAX	**	I*4	D5MIN	**	R*4	ETENDUE1
**	R*4	ETENDUE2	**	I*4	F2INC	**	I*4	F2MAX	**	I*4	F2MIN
**	I*4	F4INC	**	I*4	F4MAX	**	I*4	F4MIN	2-00000040	L*4	FIRSTFLAG
**	L*4	GOODFLAG	2-00000000	I*4	I1	**	I*4	I2	**	I*4	I3
**	I*4	I4	**	L*4	RAYFLAG	**	L*4	RAYGEO	2-0000002C	R*4	RD1
2-00000044	R*4	RD1OPT	2-00000024	R*4	RD3	2-0000004C	R*4	RD3OPT	2-0000001C	R*4	RD5
2-00000054	R*4	RD5OPT	2-00000028	R*4	RF2	2-00000048	R*4	RF2OPT	2-00000020	R*4	RF4
2-00000050	R*4	RF4OPT	2-00000004	R*4	RI	2-00000008	R*4	RI1PRIME	2-0000000C	R*4	RI2PRIME
2-00000010	R*4	RI3PRIME	2-00000030	R*4	RO	2-0000005C	R*4	RO1OPT	2-00000034	R*4	RO1PRIME
2-00000060	R*4	RO2OPT	2-00000038	R*4	RO2PRIME	2-00000064	R*4	RO3OPT	2-0000003C	R*4	RO3PRIME
2-00000014	R*4	ROCRIT	2-00000058	R*4	ROOPT	2-00000018	R*4	ROPRIMECR			

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
1-00000059	100'	1-00000066	101'	1-000000CF	102'	1-00000103	103'	1-0000012C	104'

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name	Type	Name	Type	Name	Type	Name
	COMP_D1	L*4	ETENDO1T	R*4	ETENDUE		FOR$CLOSE		FOR$OPEN		MOVEDATA
L*4	RAYSEQ	L*4	RAYSGOOD								

```
0001    c
0002    c function to determine if rays are in proper sequence
0003    c
0004    c Input data (parameters):
0005    c
0006    c  a - angle of first ray
0007    c  b - angle of second ray
0008    c  c - angle of third ray
0009    c  d - distance of rays from optical axis
0010    c
0011    c Output data (returned by function)
0012    c
0013    c  rayseq - 1=>rays properly specified
0014    c
0015    c Subprograms called: None
0016    c
0017    ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0018    c
0019          logical function rayseq(a,b,c,d)
0020    c
0021    c declarations
0022    c
0023          implicit none
0024          real a,b,c,d ! defined in header block
0025    c
0026    c computations
0027    c
0028          rayseq = (a .GT. b)
0029          rayseq = rayseq .AND. (b .GT. c)
0030          rayseq = rayseq .AND. (d .GT. 0.0)
0031          end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	57	PIC CON REL LCL SHR EXE RD NOWRT LONG
2	$LOCAL	4	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	61	

ENTRY POINTS

Address	Type	Name
0-00000000	L*4	RAYSEQ

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name	Address	Type	Name
AP-00000004@	R*4	A	AP-00000008@	R*4	B	AP-0000000C@	R*4	C	AP-00000010@	R*4	D

```
0001    c
0002    c subroutine to perform ray calculations
0003    c
0004    c This subroutine finds the spacing
0005    c between the source and first lens that will allow an image
0006    c to form with the geometry that has been passed in.
0007    c
0008    c Input Data (Parameters):
0009    c  f2 - focal length of first lens
0010    c  d3 - spacing between lenses
0011    c  f4 - focal length of second lens
0012    c  d5 - distance from second lens to image
0013    c  i1 - height of source
0014    c  i1prime - angle of first source ray
0015    c  i2prime - angle of second source ray
0016    c  i3prime - angle of third source ray
0017    c
0018    c Output Data (Parameters):
0019    c  d1 - distance from source to first lens
0020    c  io - height of image
0021    c  o1prime -angle of first image ray
0022    c  o2prime - angle of second image ray
0023    c  o3prime - angle of third image ray
0024    c
0025    c Subprograms called:
0026    c  raycalc - calculates output ray given input ray
0027    c
0028    ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0029    c
0030    c
0031          subroutine comp_d1 (f2,d3,f4,d5,d1,i1,i1prime
0032         &,i2prime,i3prime,io,o1prime,o2prime,o3prime)
0033    c declarations
0034          implicit none
0035          real f2 !focal length of first lens
0036          real d3 !distance between lenses
0037          real f4 !focal length of second lens
0038          real d5 !distance from second lens to image
0039          real d1 !distance from source to first lens
0040          real i1 !distance from source to optical axis
0041          real i1prime ! angle of 1st source ray to optical axis
0042          real i2prime ! angle of 2nd source ray to optical axis
0043          real i3prime ! angle of 3rd source ray to optical axis
0044          real io ! distance from image to optical axis
0045          real o1prime ! angle of 1st image ray to optical axis
0046          real o2prime ! angle of 2nd image ray to optical axis
0047          real o3prime ! angle of 3rd image ray to optical axis
0048          real io1 ! height of first image ray
0049          real io2 ! height of second image ray
0050          real io3 ! height of third image ray
0051          real alpha ! intermediate value
0052          real top ! intermediate value
0053          real bottom ! intermediate value
0054    c
0055    c calculate d1 with equation 5.33 from Hecht p 145
0056    c
0057          alpha = (d5*f4)/(d5-f4)
```

```
0058           top = f2*(d3-alpha)
0059           bottom = d3-f2-alpha
0060           d1 = top/bottom
0061      c
0062      c determine properties of output rays
0063      c
0064           call raycalc(d1,f2,d3,f4,d5,i1,i1prime,io1,o1prime)
0065           call raycalc(d1,f2,d3,f4,d5,i1,i2prime,io2,o2prime)
0066           call raycalc(d1,f2,d3,f4,d5,i1,i3prime,io3,o3prime)
0067           io = io1
0068      c
0069      c cleanup
0070      c
0071           end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	198	PIC CON REL LCL SHR EXE RD NOWRT LONG
2	$LOCAL	132	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	330	

ENTRY POINTS

Address	Type	Name
0-00000000		COMP_D1

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name	Address	Type	Name
**	R*4	ALPHA	**	R*4	BOTTOM	AP-00000014@	R*4	D1	AP-00000008@	R*4	D3
AP-00000010@	R*4	D5	AP-00000004@	R*4	F2	AP-0000000C@	R*4	F4	AP-00000018@	R*4	I1
AP-0000001C@	R*4	I1PRIME	AP-00000020@	R*4	I2PRIME	AP-00000024@	R*4	I3PRIME	AP-00000028@	R*4	IO
2-00000000	R*4	IO1	2-00000004	R*4	IO2	2-00000008	R*4	IO3	AP-0000002C@	R*4	O1PRIME
AP-00000030@	R*4	O2PRIME	AP-00000034@	R*4	O3PRIME	**	R*4	TOP			

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name
	RAYCALC

```
0001    c
0002    c function to determine if rays meet checking criterion
0003    c
0004    c Input data (parameters):
0005    c
0006    c  a - height of ray
0007    c  b - maximum allowable height of ray
0008    c  c - angle of first ray
0009    c  d - maximum allowable angle of ray
0010    c  e - angle of second ray
0011    c  f - angle of third ray
0012    c
0013    c Output data (returned by function)
0014    c
0015    c  raysgood - 1=>rays meet criterion
0016    c
0017    c Subprograms called: None
0018    c
0019    ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0020    c
0021          logical function raysgood(a,b,c,d,e,f)
0022    c
0023    c declarations
0024    c
0025          implicit none
0026          real a,b,c,d,e,f ! defined in header block
0027    c
0028    c computations
0029    c
0030          raysgood = (abs(a) .LE. abs(b))
0031          raysgood = raysgood .AND. (abs(c) .LE. abs(d))
0032          raysgood = raysgood .AND. (abs(e) .LE. abs(d))
0033          raysgood = raysgood .AND. (abs(f) .LE. abs(d))
0034          end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	125	PIC CON REL LCL SHR EXE RD NOWRT LONG
2	$LOCAL	4	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	129	

ENTRY POINTS

Address	Type	Name
0-00000000	L*4	RAYSGOOD

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name	Address	Type	Name
AP-00000004@	R*4	A	AP-00000008@	R*4	B	AP-0000000C@	R*4	C	AP-00000010@	R*4	D
AP-00000014@	R*4	E	AP-00000018@	R*4	F						

```
0001    c
0002    c subroutine to copy optimal data
0003    c
0004    c This subroutine copies data from the variables describing
0005    c the current raytrace to the variables describing the best raytrace
0006    c encountered so far
0007    c
0008    c Input data (parameters):
0009    c
0010    c a1 - current distance from source to first lens
0011    c a2 - current focal length of first lens
0012    c a3 - current spacing between lenses
0013    c a4 - current focal length of second lens
0014    c a5 - current distance from second lens to image
0015    c a6 - current height of image
0016    c a7 - current angle of first image ray
0017    c a8 - current angle of second image ray
0018    c a9 - current angle of third image ray
0019    c
0020    c Output data (parameters)
0021    c
0022    c b1 - best distance from source to first lens
0023    c b2 - best focal length of first lens
0024    c b3 - best spacing between lenses
0025    c b4 - best focal length of second lens
0026    c b5 - best distance from second lens to image
0027    c b6 - best height of image
0028    c b7 - best angle of first image ray
0029    c b8 - best angle of second image ray
0030    c b9 - best angle of third image ray
0031    c
0032    c Subprograms called: None
0033    c
0034    ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0035    c
0036          subroutine movedata(b1,b2,b3,b4,b5,b6,b7,b8,b9,
0037         &a1,a2,a3,a4,a5,a6,a7,a8,a9)
0038    c
0039    c declarations (defined in header block)
0040    c
0041          implicit none
0042          real a1,a2,a3,a4,a5,a6,a7,a8,a9
0043          real b1,b2,b3,b4,b5,b6,b7,b8,b9
0044    c
0045    c computations
0046    c
0047          b1=a1
0048          b2=a2
0049          b3=a3
0050          b4=a4
0051          b5=a5
0052          b6=a6
0053          b7=a7
0054          b8=a8
0055          b9=a9
0056          end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	48	PIC CON REL LCL SHR EXE RD NOWRT LONG
	Total Space Allocated	48	

ENTRY POINTS

	Address	Type	Name
0-	00000000		MOVEDATA

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name	Address	Type	Name
AP-00000028@	R*4	A1	AP-0000002C@	R*4	A2	AP-00000030@	R*4	A3	AP-00000034@	R*4	A4
AP-00000038@	R*4	A5	AP-0000003C@	R*4	A6	AP-00000040@	R*4	A7	AP-00000044@	R*4	A8
AP-00000048@	R*4	A9	AP-00000004@	R*4	B1	AP-00000008@	R*4	B2	AP-0000000C@	R*4	B3
AP-00000010@	R*4	B4	AP-00000014@	R*4	B5	AP-00000018@	R*4	B6	AP-0000001C@	R*4	B7
AP-00000020@	R*4	B8	AP-00000024@	R*4	B9						

```
0001    c
0002    c
0003    c function to determine minimum error
0004    c
0005    c Input data (parameters):
0006    c
0007    c a - height of current ray
0008    c b - angle of current ray
0009    c e - height of criterion ray
0010    c f - angle of criterion ray
0011    c c - height of best ray so far
0012    c d - angle of best ray so far
0013    c
0014    c Output data (returned by function)
0015    c
0016    c  etendoit - 1=>current ray has smallest error
0017    c
0018    c Subprograms called: None
0019    c
0020    ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0021    c
0022          logical function etendoit(a,b,e,f,c,d)
0023    c
0024    c declarations
0025    c
0026          implicit none
0027          real a,b,e,f,c,d ! defined in header block
0028          real error1,error2,error3,error4 ! intermediate values
0029    c
0030    c computations
0031    c
0032    c sum of squared relative errors for the current ray
0033    c must be less than or equal to sum of squared relative
0034    c errors for the best ray so far if the function
0035    c is to return a true flag. otherwise function returns false
0036    c
0037          error1 = (abs(a)-abs(e))/abs(e)
0038          error2 = (abs(b)-abs(f))/abs(f)
0039          error3 = (abs(c)-abs(e))/abs(e)
0040          error4 = (abs(d)-abs(f))/abs(f)
0041          error1 = error1**2
0042          error2 = error2**2
0043          error3 = error3**2
0044          error4 = error4**2
0045          error1 = error1 + error2
0046          error3 = error3 + error4
0047          etendoit = (error1 .LE. error3)
0048          end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	118	PIC CON REL LCL SHR EXE RD NOWRT LONG
2	$LOCAL	4	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	122	

ENTRY POINTS

	Address	Type	Name
0	0-00000000	L*4	ETENDOIT

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name	Address	Type	Name
AP-00000004@	R*4	A	AP-00000008@	R*4	B	AP-00000014@	R*4	C	AP-00000018@	R*4	D
AP-0000000C@	R*4	E	**	R*4	ERROR1	**	R*4	ERROR2	**	R*4	ERROR3
**	R*4	ERROR4	AP-00000010@	R*4	F						

```
0001    c
0002    c
0003    c Subroutine to calculate output ray given input ray
0004    c
0005    c This subroutine uses repeated matrix multiplication to
0006    c find an output ray as a function of an input ray
0007    c
0008    c Input data (parameters):
0009    c  md1 -  distance from source to first lens
0010    c  mf2 -  focal length of first lens
0011    c  md3 -  distance between lenses
0012    c  mf4 -  focal length of second lens
0013    c  md5 -  distance from second lens to image
0014    c  mi -   height of source ray
0015    c  miprime - angle of source ray
0016    c
0017    c Output data (paramaters):
0018    c  mo - height of image ray
0019    c  moprime - angle of image ray
0020    c
0021    c Subprograms called:
0022    c  matmult - multiplies 2X2 matrices
0023    c
0024    ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0025    c
0026          subroutine raycalc(md1,mf2,md3,mf4,md5,mi,miprime,mo,moprime)
0027    c
0028    c declarations
0029    c
0030          implicit none
0031          real md1,mf2,md3,mf4,md5,mi,miprime,mo,moprime !see header
0032          real m5(2,2)/1.,0.,0.,1./ !last space
0033          real m4(2,2)/1.,0.,0.,1./ !second lens
0034          real m3(2,2)/1.,0.,0.,1./ !middle space
0035          real m2(2,2)/1.,0.,0.,1./ !first lens
0036          real m1(2,2)/1.,0.,0.,1./ !first space
0037          real mp1(2,2),mp2(2,2),mp3(2,2),mp4(2,2) !subproducts
0038    c
0039    c write properties to matrices
0040    c
0041          m5(1,2) = md5
0042          m3(1,2) = md3
0043          m1(1,2) = md1
0044          m4(2,1) = -1.0/mf4
0045          m2(2,1) = -1.0/mf2
0046    c
0047    c matrix multiplication
0048    c
0049          call matmult(m5,m4,mp4)
0050          call matmult(mp4,m3,mp3)
0051          call matmult(mp3,m2,mp2)
0052          call matmult(mp2,m1,mp1)
0053    c
0054    c ray results
0055    c
0056          mo = mi*mp1(1,1)+miprime*mp1(1,2)
0057          moprime = mi*mp1(2,1)+miprime*mp1(2,2)
```

```
0058          end


PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                                 109   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  2 $LOCAL                                208   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                 317


ENTRY POINTS

    Address  Type  Name

  0-00000000       RAYCALC


VARIABLES

    Address  Type  Name             Address  Type  Name             Address  Type  Name             Address  Type  Name

 AP-00000004@ R*4  MD1           AP-0000000C@ R*4  MD3           AP-00000014@ R*4  MD5           AP-00000008@ R*4  MF2
 AP-00000010@ R*4  MF4           AP-00000018@ R*4  MI            AP-0000001C@ R*4  MIPRIME       AP-00000020@ R*4  MO
 AP-00000024@ R*4  MOPRIME


ARRAYS

    Address  Type  Name               Bytes  Dimensions

  2-00000040  R*4  M1                   16  (2, 2)
  2-00000030  R*4  M2                   16  (2, 2)
  2-00000020  R*4  M3                   16  (2, 2)
  2-00000010  R*4  M4                   16  (2, 2)
  2-00000000  R*4  M5                   16  (2, 2)
  2-00000050  R*4  MP1                  16  (2, 2)
  2-00000060  R*4  MP2                  16  (2, 2)
  2-00000070  R*4  MP3                  16  (2, 2)
  2-00000080  R*4  MP4                  16  (2, 2)


FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name

        MATMULT
```

```
0001    c
0002    c subroutine to multiply 2x2 matrices
0003    c
0004    c Input Data (Parameters - matrices to be multiplied)
0005    c  mm2
0006    c  mm1
0007    c
0008    c Output Data (Parameter - matrix product)
0009    c  mmp
0010    c
0011    c
0012    c mmp=mm2Xmm1
0013    c
0014    c Subprograms called: None
0015    c
0016    ccccccccccccccccccccccccccccccccccccccccccccccccccccc
0017    c
0018          subroutine matmult(mm2,mm1,mmp)
0019    c
0020    c declarations
0021    c
0022          real mm2(2,2),mm1(2,2),mmp(2,2) !see header block
0023          integer row,col,k !loop counters
0024    c
0025    c computation
0026    c
0027          do row = 1,2
0028           do col = 1,2
0029            mmp(row,col) = 0.0
0030            do k = 1,2
0031             mmp(row,col) = mmp(row,col)+mm2(row,k)*mm1(k,col)
0032            end do !k
0033           end do !col
0034          end do !row
0035          end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	118	PIC CON REL LCL SHR EXE RD NOWRT LONG
2	$LOCAL	60	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	178	

ENTRY POINTS

Address	Type	Name
0-00000000		MATMULT

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name
**	I*4	COL	**	I*4	K	**	I*4	ROW

ARRAYS

Address	Type	Name	Bytes	Dimensions
AP-00000008@	R*4	MM1	16	(2, 2)
AP-00000004@	R*4	MM2	16	(2, 2)
AP-0000000C@	R*4	MMP	16	(2, 2)

```
0001    c
0002    c function to calculate etendue
0003    c
0004    c Input data (Parameters):
0005    c
0006    c  r = radius
0007    c  na = numerical aperture
0008    c
0009    c Output Data (Returned in function call):
0010    c
0011    c etendue = light gathering power
0012    c
0013    c Subprograms called: None
0014    c
0015    ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0016    c
0017          real function etendue(r,na)
0018    c
0019    c declarations
0020    c
0021          real r,na !defined in header
0022          real pi ! intermediate data
0023    c
0024    c computation
0025    c
0026          pi = 4.*atan(1.)
0027          etendue = (pi*r*na)**2
0028    c
0029    c cleanup
0030    c
0031          end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	33	PIC CON REL LCL SHR EXE RD NOWRT LONG
2	$LOCAL	4	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	37	

ENTRY POINTS

Address	Type	Name
0-00000000	R*4	ETENDUE

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name
AP-00000008@	R*4	NA	**	R*4	PI	AP-00000004@	R*4	R

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name
R*4	MTH$ATAN

COMMAND QUALIFIERS

```
FORTRAN RAY/LIST

/CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/SHOW=(NODICTIONARY,NOINCLUDE,MAP,NOPREPROCESSOR,SINGLE)
/STANDARD=(NOSEMANTIC,NOSOURCE_FORM,NOSYNTAX)
/WARNINGS=(NODECLARATIONS,GENERAL,NOULTRIX,NOVAXELN)
/CONTINUATIONS=19  /NOCROSS_REFERENCE  /NOD_LINES  /NOEXTEND_SOURCE
/F77  /NOG_FLOATING  /I4  /NOMACHINE_CODE  /OPTIMIZE  /NOPARALLEL
/NOANALYSIS_DATA
/NODIAGNOSTICS
/LIST=TMP$WORK1:[SPACE1.SKERCEL]RAY.LIS;1
/OBJECT=TMP$WORK1:[SPACE1.SKERCEL]RAY.OBJ;1
```

COMPILATION STATISTICS

```
Run Time:           1.37 seconds
Elapsed Time:       3.21 seconds
Page Faults:        776
Dynamic Memory:     430 pages
```

```
University of Tennessee VAXcluster member UTKVX3

Username: SKERCEL
Password:
        Welcome to VAX/VMS version V5.2 on node UTKVX3
     Last interactive login on Thursday,  8-FEB-1990 14:09
        Group 14143 User 133208 logged in Thursday,  8-FEB-1990 14:26
1m
[08-Feb-90] DCA Network Disconnect Character to change on Feb. 12, 1990
[07-Jan-90] JNET Version 3.4 installed on VAXcluster
[20-Dec-89] KERMIT default parity set to SPACE
[14-Dec-89] UTCC Spring Short Courses Schedule online. Type HELP SHORT_COURSES.
[22-Nov-89] I-DEAS Plotting Capabilities Enhanced
[21-Nov-89] GCG Sequence Analysis Software Package available on VAXcluster
[16-Nov-89] ARC/INFO Version 5.0 installed
[19-Nov-89] UTCC Changed Its Network Parity on Sunday November 19th
[15-Nov-89] Notice of VAX2 and VAX4 Unavailability, Mon.-Sat.,6-6:30am
[02-Nov-89] PRECAL supports color for Tektronix 41xx and 42xx terminals
[02-Nov-89] ACSL upgraded from Level 8R to Level 98
[30-Oct-89] VAXcluster now open to users 4am-8am Monday-Saturday
[29-Oct-89] DCA destination VAX selects either VAX2 or VAX4

%NOTICE-I-MORE, Type "NOTICE" for full display of notices
0m

62"p F

Where do you want your printouts? Enter remote number: 2
$ set default tmp$work1:[space1.skercel]
$ show default
  TMP$WORK1:[SPACE1.SKERCEL]
$ fortran ray/list
$ link ray
$ run ray
$ type goodray.dat
Input Ray Height:  0.800 Ray Angles:  0.2250  0.0000 -0.2250
Etendue of Source:  0.320 Etendue of Target:  0.391
```

D1	F2	D3	F4	D5	Height	Angle 1	Angle 2	Angle 3
28.337	75.000	106.000	100.000	294.000	-2.495	-0.080	-0.007	0.065
27.739	75.000	107.000	100.000	296.000	-2.488	-0.080	-0.007	0.065
27.636	75.000	107.000	100.000	297.000	-2.496	-0.079	-0.007	0.065
27.026	75.000	108.000	100.000	299.000	-2.489	-0.079	-0.007	0.065
26.923	75.000	108.000	100.000	300.000	-2.496	-0.079	-0.007	0.065
26.404	75.000	109.000	100.000	301.000	-2.482	-0.080	-0.007	0.065
26.301	75.000	109.000	100.000	302.000	-2.489	-0.079	-0.007	0.065
26.198	75.000	109.000	100.000	303.000	-2.496	-0.079	-0.007	0.065
25.666	75.000	110.000	100.000	304.000	-2.481	-0.079	-0.007	0.066
25.563	75.000	110.000	100.000	305.000	-2.488	-0.079	-0.007	0.065
25.460	75.000	110.000	100.000	306.000	-2.495	-0.079	-0.007	0.065
24.915	75.000	111.000	100.000	307.000	-2.480	-0.079	-0.007	0.066
24.811	75.000	111.000	100.000	308.000	-2.487	-0.079	-0.007	0.066
24.708	75.000	111.000	100.000	309.000	-2.493	-0.079	-0.007	0.065
24.254	75.000	112.000	100.000	309.000	-2.471	-0.080	-0.007	0.066
24.150	75.000	112.000	100.000	310.000	-2.478	-0.079	-0.007	0.066
24.046	75.000	112.000	100.000	311.000	-2.485	-0.079	-0.007	0.066
23.942	75.000	112.000	100.000	312.000	-2.491	-0.079	-0.007	0.066
23.475	75.000	113.000	100.000	312.000	-2.469	-0.080	-0.007	0.066
23.840	75.000	112.000	100.000	313.000	-2.498	-0.079	-0.007	0.065
23.370	75.000	113.000	100.000	313.000	-2.475	-0.079	-0.007	0.066

23.266	75.000	113.000	100.000	314.000	-2.482	-0.079	-0.007	0.066
23.162	75.000	113.000	100.000	315.000	-2.489	-0.079	-0.007	0.066
22.680	75.000	114.000	100.000	315.000	-2.466	-0.080	-0.007	0.066
23.059	75.000	113.000	100.000	316.000	-2.495	-0.079	-0.007	0.066
22.575	75.000	114.000	100.000	316.000	-2.472	-0.079	-0.007	0.066
22.471	75.000	114.000	100.000	317.000	-2.479	-0.079	-0.007	0.066
22.367	75.000	114.000	100.000	318.000	-2.485	-0.079	-0.007	0.066
21.870	75.000	115.000	100.000	318.000	-2.462	-0.080	-0.006	0.067
22.263	75.000	114.000	100.000	319.000	-2.492	-0.079	-0.007	0.066
21.764	75.000	115.000	100.000	319.000	-2.468	-0.079	-0.006	0.067
22.161	75.000	114.000	100.000	320.000	-2.498	-0.079	-0.007	0.066
21.659	75.000	115.000	100.000	320.000	-2.475	-0.079	-0.006	0.066

Minimum Error of Angle 1 and Height:

D1	F2	D3	F4	D5	Height	Angle 1	Angle 2	Angle 3
28.925	75.000	105.000	100.000	292.000	-2.500	-0.079	-0.007	0.065

$ type allray.dat

Input Ray Height: 0.800 Ray Angles: 0.2250 0.0000 -0.2250

D1	F2	D3	F4	D5	Height	Angle 1	Angle 2	Angle 3
30.928	75.000	100.000	100.000	290.000	-2.587	-0.078	-0.008	0.062
30.580	75.000	101.000	100.000	290.000	-2.566	-0.078	-0.008	0.062
30.226	75.000	102.000	100.000	290.000	-2.546	-0.078	-0.008	0.063
29.867	75.000	103.000	100.000	290.000	-2.526	-0.079	-0.008	0.064
29.502	75.000	104.000	100.000	290.000	-2.506	-0.079	-0.008	0.064
29.131	75.000	105.000	100.000	290.000	-2.485	-0.080	-0.007	0.065
28.754	75.000	106.000	100.000	290.000	-2.465	-0.080	-0.007	0.066
28.370	75.000	107.000	100.000	290.000	-2.445	-0.081	-0.007	0.066
27.981	75.000	108.000	100.000	290.000	-2.425	-0.081	-0.007	0.067
27.584	75.000	109.000	100.000	290.000	-2.404	-0.082	-0.007	0.068
27.181	75.000	110.000	100.000	290.000	-2.384	-0.082	-0.007	0.069
26.771	75.000	111.000	100.000	290.000	-2.364	-0.083	-0.007	0.069
26.354	75.000	112.000	100.000	290.000	-2.343	-0.084	-0.007	0.070
25.930	75.000	113.000	100.000	290.000	-2.323	-0.084	-0.007	0.071
25.498	75.000	114.000	100.000	290.000	-2.303	-0.085	-0.007	0.072
25.058	75.000	115.000	100.000	290.000	-2.283	-0.085	-0.006	0.072
24.611	75.000	116.000	100.000	290.000	-2.262	-0.086	-0.006	0.073
24.156	75.000	117.000	100.000	290.000	-2.242	-0.086	-0.006	0.074
23.692	75.000	118.000	100.000	290.000	-2.222	-0.087	-0.006	0.075
23.219	75.000	119.000	100.000	290.000	-2.202	-0.088	-0.006	0.076
22.738	75.000	120.000	100.000	290.000	-2.181	-0.088	-0.006	0.077
30.832	75.000	100.000	100.000	291.000	-2.595	-0.077	-0.008	0.061
30.483	75.000	101.000	100.000	291.000	-2.574	-0.078	-0.008	0.062
30.128	75.000	102.000	100.000	291.000	-2.554	-0.078	-0.008	0.063
29.767	75.000	103.000	100.000	291.000	-2.534	-0.079	-0.008	0.063
29.400	75.000	104.000	100.000	291.000	-2.513	-0.079	-0.008	0.064
29.028	75.000	105.000	100.000	291.000	-2.493	-0.080	-0.007	0.065
28.649	75.000	106.000	100.000	291.000	-2.472	-0.080	-0.007	0.065
28.264	75.000	107.000	100.000	291.000	-2.452	-0.081	-0.007	0.066
27.872	75.000	108.000	100.000	291.000	-2.432	-0.081	-0.007	0.067
27.474	75.000	109.000	100.000	291.000	-2.411	-0.082	-0.007	0.068
27.069	75.000	110.000	100.000	291.000	-2.391	-0.082	-0.007	0.068
26.657	75.000	111.000	100.000	291.000	-2.371	-0.083	-0.007	0.069
26.238	75.000	112.000	100.000	291.000	-2.350	-0.083	-0.007	0.070
25.812	75.000	113.000	100.000	291.000	-2.330	-0.084	-0.007	0.071
25.378	75.000	114.000	100.000	291.000	-2.309	-0.084	-0.007	0.071
24.936	75.000	115.000	100.000	291.000	-2.289	-0.085	-0.006	0.072
24.486	75.000	116.000	100.000	291.000	-2.269	-0.086	-0.006	0.073
24.029	75.000	117.000	100.000	291.000	-2.248	-0.086	-0.006	0.074
23.563	75.000	118.000	100.000	291.000	-2.228	-0.087	-0.006	0.075
23.088	75.000	119.000	100.000	291.000	-2.208	-0.088	-0.006	0.076
22.604	75.000	120.000	100.000	291.000	-2.187	-0.088	-0.006	0.076

30.738	75.000	100.000	100.000	292.000	-2.603	-0.077	-0.008	0.061
30.387	75.000	101.000	100.000	292.000	-2.582	-0.078	-0.008	0.062
30.030	75.000	102.000	100.000	292.000	-2.562	-0.078	-0.008	0.062
29.668	75.000	103.000	100.000	292.000	-2.541	-0.079	-0.008	0.063
29.299	75.000	104.000	100.000	292.000	-2.521	-0.079	-0.008	0.064
28.925	75.000	105.000	100.000	292.000	-2.500	-0.079	-0.007	0.065
28.544	75.000	106.000	100.000	292.000	-2.480	-0.080	-0.007	0.065
28.158	75.000	107.000	100.000	292.000	-2.459	-0.080	-0.007	0.066
27.764	75.000	108.000	100.000	292.000	-2.439	-0.081	-0.007	0.067
27.364	75.000	109.000	100.000	292.000	-2.418	-0.081	-0.007	0.067
26.957	75.000	110.000	100.000	292.000	-2.398	-0.082	-0.007	0.068
26.543	75.000	111.000	100.000	292.000	-2.377	-0.083	-0.007	0.069
26.122	75.000	112.000	100.000	292.000	-2.357	-0.083	-0.007	0.070
25.694	75.000	113.000	100.000	292.000	-2.336	-0.084	-0.007	0.070
25.258	75.000	114.000	100.000	292.000	-2.316	-0.084	-0.007	0.071
24.814	75.000	115.000	100.000	292.000	-2.295	-0.085	-0.006	0.072
24.362	75.000	116.000	100.000	292.000	-2.275	-0.085	-0.006	0.073
23.902	75.000	117.000	100.000	292.000	-2.255	-0.086	-0.006	0.074
23.434	75.000	118.000	100.000	292.000	-2.234	-0.087	-0.006	0.074
22.957	75.000	119.000	100.000	292.000	-2.214	-0.087	-0.006	0.075
22.471	75.000	120.000	100.000	292.000	-2.193	-0.088	-0.006	0.076
30.644	75.000	100.000	100.000	293.000	-2.611	-0.077	-0.008	0.061
30.291	75.000	101.000	100.000	293.000	-2.590	-0.077	-0.008	0.062
29.933	75.000	102.000	100.000	293.000	-2.569	-0.078	-0.008	0.062
29.569	75.000	103.000	100.000	293.000	-2.549	-0.078	-0.008	0.063
29.199	75.000	104.000	100.000	293.000	-2.528	-0.079	-0.008	0.064
28.823	75.000	105.000	100.000	293.000	-2.508	-0.079	-0.007	0.064
28.441	75.000	106.000	100.000	293.000	-2.487	-0.080	-0.007	0.065
28.052	75.000	107.000	100.000	293.000	-2.467	-0.080	-0.007	0.066
27.657	75.000	108.000	100.000	293.000	-2.446	-0.081	-0.007	0.066
27.255	75.000	109.000	100.000	293.000	-2.425	-0.081	-0.007	0.067
26.846	75.000	110.000	100.000	293.000	-2.405	-0.082	-0.007	0.068
26.431	75.000	111.000	100.000	293.000	-2.384	-0.082	-0.007	0.069
26.007	75.000	112.000	100.000	293.000	-2.364	-0.083	-0.007	0.069
25.577	75.000	113.000	100.000	293.000	-2.343	-0.083	-0.007	0.070
25.139	75.000	114.000	100.000	293.000	-2.322	-0.084	-0.007	0.071
24.693	75.000	115.000	100.000	293.000	-2.302	-0.085	-0.006	0.072
24.239	75.000	116.000	100.000	293.000	-2.281	-0.085	-0.006	0.073
23.777	75.000	117.000	100.000	293.000	-2.261	-0.086	-0.006	0.073
23.306	75.000	118.000	100.000	293.000	-2.240	-0.086	-0.006	0.074
22.827	75.000	119.000	100.000	293.000	-2.220	-0.087	-0.006	0.075
22.338	75.000	120.000	100.000	293.000	-2.199	-0.088	-0.006	0.076
30.550	75.000	100.000	100.000	294.000	-2.619	-0.077	-0.008	0.061
30.196	75.000	101.000	100.000	294.000	-2.598	-0.077	-0.008	0.061
29.836	75.000	102.000	100.000	294.000	-2.577	-0.078	-0.008	0.062
29.471	75.000	103.000	100.000	294.000	-2.557	-0.078	-0.008	0.063
29.099	75.000	104.000	100.000	294.000	-2.536	-0.079	-0.008	0.063
28.721	75.000	105.000	100.000	294.000	-2.515	-0.079	-0.007	0.064
28.337	75.000	106.000	100.000	294.000	-2.495	-0.080	-0.007	0.065
27.947	75.000	107.000	100.000	294.000	-2.474	-0.080	-0.007	0.066
27.550	75.000	108.000	100.000	294.000	-2.453	-0.081	-0.007	0.066
27.147	75.000	109.000	100.000	294.000	-2.432	-0.081	-0.007	0.067
26.736	75.000	110.000	100.000	294.000	-2.412	-0.082	-0.007	0.068
26.318	75.000	111.000	100.000	294.000	-2.391	-0.082	-0.007	0.068
25.893	75.000	112.000	100.000	294.000	-2.370	-0.083	-0.007	0.069
25.461	75.000	113.000	100.000	294.000	-2.350	-0.083	-0.007	0.070
25.021	75.000	114.000	100.000	294.000	-2.329	-0.084	-0.007	0.071
24.573	75.000	115.000	100.000	294.000	-2.308	-0.084	-0.006	0.072
24.116	75.000	116.000	100.000	294.000	-2.288	-0.085	-0.006	0.072
23.652	75.000	117.000	100.000	294.000	-2.267	-0.086	-0.006	0.073

23.179	75.000	118.000	100.000	294.000	-2.246	-0.086	-0.006	0.074
22.697	75.000	119.000	100.000	294.000	-2.225	-0.087	-0.006	0.075
22.206	75.000	120.000	100.000	294.000	-2.205	-0.088	-0.006	0.076
30.457	75.000	100.000	100.000	295.000	-2.627	-0.077	-0.008	0.061
30.101	75.000	101.000	100.000	295.000	-2.606	-0.077	-0.008	0.061
29.740	75.000	102.000	100.000	295.000	-2.585	-0.077	-0.008	0.062
29.373	75.000	103.000	100.000	295.000	-2.564	-0.078	-0.008	0.063
29.000	75.000	104.000	100.000	295.000	-2.543	-0.078	-0.008	0.063
28.621	75.000	105.000	100.000	295.000	-2.523	-0.079	-0.007	0.064
28.235	75.000	106.000	100.000	295.000	-2.502	-0.079	-0.007	0.065
27.843	75.000	107.000	100.000	295.000	-2.481	-0.080	-0.007	0.065
27.444	75.000	108.000	100.000	295.000	-2.460	-0.080	-0.007	0.066
27.039	75.000	109.000	100.000	295.000	-2.439	-0.081	-0.007	0.067
26.626	75.000	110.000	100.000	295.000	-2.419	-0.081	-0.007	0.067
26.207	75.000	111.000	100.000	295.000	-2.398	-0.082	-0.007	0.068
25.780	75.000	112.000	100.000	295.000	-2.377	-0.082	-0.007	0.069
25.345	75.000	113.000	100.000	295.000	-2.356	-0.083	-0.007	0.070
24.903	75.000	114.000	100.000	295.000	-2.335	-0.084	-0.007	0.071
24.453	75.000	115.000	100.000	295.000	-2.315	-0.084	-0.006	0.071
23.994	75.000	116.000	100.000	295.000	-2.294	-0.085	-0.006	0.072
23.528	75.000	117.000	100.000	295.000	-2.273	-0.085	-0.006	0.073
23.052	75.000	118.000	100.000	295.000	-2.252	-0.086	-0.006	0.074
22.568	75.000	119.000	100.000	295.000	-2.231	-0.087	-0.006	0.075
22.075	75.000	120.000	100.000	295.000	-2.211	-0.087	-0.006	0.076
30.364	75.000	100.000	100.000	296.000	-2.635	-0.076	-0.008	0.060
30.007	75.000	101.000	100.000	296.000	-2.614	-0.077	-0.008	0.061
29.645	75.000	102.000	100.000	296.000	-2.593	-0.077	-0.008	0.062
29.276	75.000	103.000	100.000	296.000	-2.572	-0.078	-0.008	0.062
28.901	75.000	104.000	100.000	296.000	-2.551	-0.078	-0.008	0.063
28.520	75.000	105.000	100.000	296.000	-2.530	-0.079	-0.007	0.064
28.133	75.000	106.000	100.000	296.000	-2.509	-0.079	-0.007	0.064
27.739	75.000	107.000	100.000	296.000	-2.488	-0.080	-0.007	0.065
27.339	75.000	108.000	100.000	296.000	-2.467	-0.080	-0.007	0.066
26.931	000	109.000	100.000	296.000	-2.447	-0.081	-0.007	0.067
26.517	75.000	110.000	100.000	296.000	-2.426	-0.081	-0.007	0.067
26.096	75.000	111.000	100.000	296.000	-2.405	-0.082	-0.007	0.068
25.667	75.000	112.000	100.000	296.000	-2.384	-0.082	-0.007	0.069
25.230	75.000	113.000	100.000	296.000	-2.363	-0.083	-0.007	0.070
24.786	75.000	114.000	100.000	296.000	-2.342	-0.083	-0.007	0.070
24.334	75.000	115.000	100.000	296.000	-2.321	-0.084	-0.006	0.071
23.873	75.000	116.000	100.000	296.000	-2.300	-0.085	-0.006	0.072
23.404	75.000	117.000	100.000	296.000	-2.279	-0.085	-0.006	0.073
22.927	75.000	118.000	100.000	296.000	-2.258	-0.086	-0.006	0.074
22.440	75.000	119.000	100.000	296.000	-2.237	-0.086	-0.006	0.074
21.944	75.000	120.000	100.000	296.000	-2.217	-0.087	-0.006	0.075
30.272	75.000	100.000	100.000	297.000	-2.643	-0.076	-0.008	0.060
29.914	75.000	101.000	100.000	297.000	-2.622	-0.077	-0.008	0.061
29.550	75.000	102.000	100.000	297.000	-2.601	-0.077	-0.008	0.061
29.179	75.000	103.000	100.000	297.000	-2.580	-0.077	-0.008	0.062
28.803	75.000	104.000	100.000	297.000	-2.559	-0.078	-0.008	0.063
28.421	75.000	105.000	100.000	297.000	-2.538	-0.078	-0.007	0.063
28.032	75.000	106.000	100.000	297.000	-2.517	-0.079	-0.007	0.064
27.636	75.000	107.000	100.000	297.000	-2.496	-0.079	-0.007	0.065
27.234	75.000	108.000	100.000	297.000	-2.475	-0.080	-0.007	0.066
26.825	75.000	109.000	100.000	297.000	-2.454	-0.080	-0.007	0.066
26.409	75.000	110.000	100.000	297.000	-2.433	-0.081	-0.007	0.067
25.985	75.000	111.000	100.000	297.000	-2.412	-0.081	-0.007	0.068
25.554	75.000	112.000	100.000	297.000	-2.391	-0.082	-0.007	0.069
25.116	75.000	113.000	100.000	297.000	-2.369	-0.083	-0.007	0.069
24.670	75.000	114.000	100.000	297.000	-2.348	-0.083	-0.007	0.070

24.215	75.000	115.000	100.000	297.000	-2.327	-0.084	-0.006	0.071
23.752	75.000	116.000	100.000	297.000	-2.306	-0.084	-0.006	0.072
23.281	75.000	117.000	100.000	297.000	-2.285	-0.085	-0.006	0.073
22.801	75.000	118.000	100.000	297.000	-2.264	-0.086	-0.006	0.073
22.312	75.000	119.000	100.000	297.000	-2.243	-0.086	-0.006	0.074
21.814	75.000	120.000	100.000	297.000	-2.222	-0.087	-0.006	0.075
30.181	75.000	100.000	100.000	298.000	-2.651	-0.076	-0.008	0.060
29.821	75.000	101.000	100.000	298.000	-2.630	-0.076	-0.008	0.061
29.455	75.000	102.000	100.000	298.000	-2.608	-0.077	-0.008	0.061
29.084	75.000	103.000	100.000	298.000	-2.587	-0.077	-0.008	0.062
28.706	75.000	104.000	100.000	298.000	-2.566	-0.078	-0.008	0.063
28.321	75.000	105.000	100.000	298.000	-2.545	-0.078	-0.007	0.063
27.931	75.000	106.000	100.000	298.000	-2.524	-0.079	-0.007	0.064
27.534	75.000	107.000	100.000	298.000	-2.503	-0.079	-0.007	0.065
27.130	75.000	108.000	100.000	298.000	-2.482	-0.080	-0.007	0.065
26.719	75.000	109.000	100.000	298.000	-2.461	-0.080	-0.007	0.066
26.301	75.000	110.000	100.000	298.000	-2.439	-0.081	-0.007	0.067
25.876	75.000	111.000	100.000	298.000	-2.418	-0.081	-0.007	0.068
25.443	75.000	112.000	100.000	298.000	-2.397	-0.082	-0.007	0.068
25.002	75.000	113.000	100.000	298.000	-2.376	-0.082	-0.007	0.069
24.554	75.000	114.000	100.000	298.000	-2.355	-0.083	-0.007	0.070
24.097	75.000	115.000	100.000	298.000	-2.334	-0.084	-0.006	0.071
23.633	75.000	116.000	100.000	298.000	-2.313	-0.084	-0.006	0.072
23.159	75.000	117.000	100.000	298.000	-2.292	-0.085	-0.006	0.072
22.677	75.000	118.000	100.000	298.000	-2.271	-0.085	-0.006	0.073
22.186	75.000	119.000	100.000	298.000	-2.249	-0.086	-0.006	0.074
21.685	75.000	120.000	100.000	298.000	-2.228	-0.087	-0.006	0.075
30.090	75.000	100.000	100.000	299.000	-2.659	-0.076	-0.008	0.060
29.729	75.000	101.000	100.000	299.000	-2.637	-0.076	-0.008	0.060
29.362	75.000	102.000	100.000	299.000	-2.616	-0.077	-0.008	0.061
28.988	75.000	103.000	100.000	299.000	-2.595	-0.077	-0.008	0.062
28.609	75.000	104.000	100.000	299.000	-2.574	-0.078	-0.008	0.062
28.223	75.000	105.000	100.000	299.000	-2.553	-0.078	-0.007	0.063
27.831	75.000	106.000	100.000	299.000	-2.531	-0.078	-0.007	0.064
27.432	75.000	107.000	100.000	299.000	-2.510	-0.079	-0.007	0.064
27.026	75.000	108.000	100.000	299.000	-2.489	-0.079	-0.007	0.065
26.613	75.000	109.000	100.000	299.000	-2.468	-0.080	-0.007	0.066
26.194	75.000	110.000	100.000	299.000	-2.446	-0.081	-0.007	0.067
25.766	75.000	111.000	100.000	299.000	-2.425	-0.081	-0.007	0.067
25.332	75.000	112.000	100.000	299.000	-2.404	-0.082	-0.007	0.068
24.889	75.000	113.000	100.000	299.000	-2.383	-0.082	-0.007	0.069
24.439	75.000	114.000	100.000	299.000	-2.361	-0.083	-0.007	0.070
23.980	75.000	115.000	100.000	299.000	-2.340	-0.083	-0.006	0.071
23.513	75.000	116.000	100.000	299.000	-2.319	-0.084	-0.006	0.071
23.038	75.000	117.000	100.000	299.000	-2.298	-0.085	-0.006	0.072
22.553	75.000	118.000	100.000	299.000	-2.277	-0.085	-0.006	0.073
22.059	75.000	119.000	100.000	299.000	-2.255	-0.086	-0.006	0.074
21.556	75.000	120.000	100.000	299.000	-2.234	-0.086	-0.006	0.075
30.000	75.000	100.000	100.000	300.000	-2.667	-0.075	-0.008	0.059
29.637	75.000	101.000	100.000	300.000	-2.645	-0.076	-0.008	0.060
29.268	75.000	102.000	100.000	300.000	-2.624	-0.076	-0.008	0.061
28.893	75.000	103.000	100.000	300.000	-2.603	-0.077	-0.008	0.061
28.512	75.000	104.000	100.000	300.000	-2.581	-0.077	-0.008	0.062
28.125	75.000	105.000	100.000	300.000	-2.560	-0.078	-0.007	0.063
27.731	75.000	106.000	100.000	300.000	-2.539	-0.078	-0.007	0.064
27.331	75.000	107.000	100.000	300.000	-2.517	-0.079	-0.007	0.064
26.923	75.000	108.000	100.000	300.000	-2.496	-0.079	-0.007	0.065
26.509	75.000	109.000	100.000	300.000	-2.475	-0.080	-0.007	0.066
26.087	75.000	110.000	100.000	300.000	-2.453	-0.080	-0.007	0.066
25.658	75.000	111.000	100.000	300.000	-2.432	-0.081	-0.007	0.067

25.221	75.000	112.000	100.000	300.000	-2.411	-0.081	-0.007	0.068
24.777	75.000	113.000	100.000	300.000	-2.389	-0.082	-0.007	0.069
24.324	75.000	114.000	100.000	300.000	-2.368	-0.083	-0.007	0.070
23.864	75.000	115.000	100.000	300.000	-2.347	-0.083	-0.006	0.070
23.394	75.000	116.000	100.000	300.000	-2.325	-0.084	-0.006	0.071
22.917	75.000	117.000	100.000	300.000	-2.304	-0.084	-0.006	0.072
22.430	75.000	118.000	100.000	300.000	-2.283	-0.085	-0.006	0.073
21.934	75.000	119.000	100.000	300.000	-2.261	-0.086	-0.006	0.074
21.429	75.000	120.000	100.000	300.000	-2.240	-0.086	-0.006	0.074
29.910	75.000	100.000	100.000	301.000	-2.675	-0.075	-0.008	0.059
29.546	75.000	101.000	100.000	301.000	-2.653	-0.076	-0.008	0.060
29.176	75.000	102.000	100.000	301.000	-2.632	-0.076	-0.008	0.061
28.799	75.000	103.000	100.000	301.000	-2.610	-0.077	-0.008	0.061
28.417	75.000	104.000	100.000	301.000	-2.589	-0.077	-0.008	0.062
28.028	75.000	105.000	100.000	301.000	-2.567	-0.078	-0.007	0.063
27.632	75.000	106.000	100.000	301.000	-2.546	-0.078	-0.007	0.063
27.230	75.000	107.000	100.000	301.000	-2.525	-0.079	-0.007	0.064
26.821	75.000	108.000	100.000	301.000	-2.503	-0.079	-0.007	0.065
26.404	75.000	109.000	100.000	301.000	-2.482	-0.080	-0.007	0.065
25.981	75.000	110.000	100.000	301.000	-2.460	-0.080	-0.007	0.066
25.550	75.000	111.000	100.000	301.000	-2.439	-0.081	-0.007	0.067
25.111	75.000	112.000	100.000	301.000	-2.417	-0.081	-0.007	0.068
24.665	75.000	113.000	100.000	301.000	-2.396	-0.082	-0.007	0.069
24.211	75.000	114.000	100.000	301.000	-2.375	-0.082	-0.007	0.069
23.748	75.000	115.000	100.000	301.000	-2.353	-0.083	-0.006	0.070
23.276	75.000	116.000	100.000	301.000	-2.332	-0.083	-0.006	0.071
22.796	75.000	117.000	100.000	301.000	-2.310	-0.084	-0.006	0.072
22.307	75.000	118.000	100.000	301.000	-2.289	-0.085	-0.006	0.073
21.809	75.000	119.000	100.000	301.000	-2.267	-0.085	-0.006	0.073
21.301	75.000	120.000	100.000	301.000	-2.246	-0.086	-0.006	0.074
29.821	75.000	100.000	100.000	302.000	-2.683	-0.075	-0.008	0.059
29.455	75.000	101.000	100.000	302.000	-2.661	-0.076	-0.008	0.060
29.083	75.000	102.000	100.000	302.000	-2.640	-0.076	-0.008	0.060
28.706	75.000	103.000	100.000	302.000	-2.618	-0.076	-0.008	0.061
28.321	75.000	104.000	100.000	302.000	-2.596	-0.077	-0.008	0.062
27.931	75.000	105.000	100.000	302.000	-2.575	-0.077	-0.007	0.062
27.534	75.000	106.000	100.000	302.000	-2.553	-0.078	-0.007	0.063
27.130	75.000	107.000	100.000	302.000	-2.532	-0.078	-0.007	0.064
26.719	75.000	108.000	100.000	302.000	-2.510	-0.079	-0.007	0.065
26.301	75.000	109.000	100.000	302.000	-2.489	-0.079	-0.007	0.065
25.875	75.000	110.000	100.000	302.000	-2.467	-0.080	-0.007	0.066
25.443	75.000	111.000	100.000	302.000	-2.446	-0.080	-0.007	0.067
25.002	75.000	112.000	100.000	302.000	-2.424	-0.081	-0.007	0.068
24.554	75.000	113.000	100.000	302.000	-2.403	-0.082	-0.007	0.068
24.097	75.000	114.000	100.000	302.000	-2.381	-0.082	-0.007	0.069
23.632	75.000	115.000	100.000	302.000	-2.359	-0.083	-0.006	0.070
23.159	75.000	116.000	100.000	302.000	-2.338	-0.083	-0.006	0.071
22.677	75.000	117.000	100.000	302.000	-2.316	-0.084	-0.006	0.072
22.186	75.000	118.000	100.000	302.000	-2.295	-0.085	-0.006	0.072
21.685	75.000	119.000	100.000	302.000	-2.273	-0.085	-0.006	0.073
21.175	75.000	120.000	100.000	302.000	-2.252	-0.086	-0.006	0.074
29.732	75.000	100.000	100.000	303.000	-2.691	-0.075	-0.008	0.059
29.365	75.000	101.000	100.000	303.000	-2.669	-0.075	-0.008	0.060
28.992	75.000	102.000	100.000	303.000	-2.647	-0.076	-0.008	0.060
28.612	75.000	103.000	100.000	303.000	-2.626	-0.076	-0.008	0.061
28.227	75.000	104.000	100.000	303.000	-2.604	-0.077	-0.008	0.062
27.835	75.000	105.000	100.000	303.000	-2.582	-0.077	-0.007	0.062
27.436	75.000	106.000	100.000	303.000	-2.561	-0.078	-0.007	0.063
27.030	75.000	107.000	100.000	303.000	-2.539	-0.078	-0.007	0.064
26.618	75.000	108.000	100.000	303.000	-2.517	-0.079	-0.007	0.064

26.198	75.000	109.000	100.000	303.000	-2.496	-0.079	-0.007	0.065
25.771	75.000	110.000	100.000	303.000	-2.474	-0.080	-0.007	0.066
25.336	75.000	111.000	100.000	303.000	-2.452	-0.080	-0.007	0.067
24.894	75.000	112.000	100.000	303.000	-2.431	-0.081	-0.007	0.067
24.443	75.000	113.000	100.000	303.000	-2.409	-0.081	-0.007	0.068
23.985	75.000	114.000	100.000	303.000	-2.388	-0.082	-0.007	0.069
23.518	75.000	115.000	100.000	303.000	-2.366	-0.082	-0.006	0.070
23.042	75.000	116.000	100.000	303.000	-2.344	-0.083	-0.006	0.070
22.558	75.000	117.000	100.000	303.000	-2.323	-0.084	-0.006	0.071
22.064	75.000	118.000	100.000	303.000	-2.301	-0.084	-0.006	0.072
21.561	75.000	119.000	100.000	303.000	-2.279	-0.085	-0.006	0.073
21.049	75.000	120.000	100.000	303.000	-2.258	-0.086	-0.006	0.074
29.644	75.000	100.000	100.000	304.000	-2.699	-0.075	-0.008	0.059
29.276	75.000	101.000	100.000	304.000	-2.677	-0.075	-0.008	0.059
28.901	75.000	102.000	100.000	304.000	-2.655	-0.076	-0.008	0.060
28.520	75.000	103.000	100.000	304.000	-2.633	-0.076	-0.008	0.061
28.133	75.000	104.000	100.000	304.000	-2.612	-0.076	-0.008	0.061
27.739	75.000	105.000	100.000	304.000	-2.590	-0.077	-0.007	0.062
27.338	75.000	106.000	100.000	304.000	-2.568	-0.077	-0.007	0.063
26.931	75.000	107.000	100.000	304.000	-2.546	-0.078	-0.007	0.063
26.517	75.000	108.000	100.000	304.000	-2.525	-0.078	-0.007	0.064
26.095	75.000	109.000	100.000	304.000	-2.503	-0.079	-0.007	0.065
25.666	75.000	110.000	100.000	304.000	-2.481	-0.079	-0.007	0.066
25.230	75.000	111.000	100.000	304.000	-2.459	-0.080	-0.007	0.066
24.786	75.000	112.000	100.000	304.000	-2.438	-0.081	-0.007	0.067
24.333	75.000	113.000	100.000	304.000	-2.416	-0.081	-0.007	0.068
23.873	75.000	114.000	100.000	304.000	-2.394	-0.082	-0.007	0.069
23.404	75.000	115.000	100.000	304.000	-2.372	-0.082	-0.006	0.069
22.926	75.000	116.000	100.000	304.000	-2.351	-0.083	-0.006	0.070
22.440	75.000	117.000	100.000	304.000	-2.329	-0.083	-0.006	0.071
21.944	75.000	118.000	100.000	304.000	-2.307	-0.084	-0.006	0.072
21.439	75.000	119.000	100.000	304.000	-2.285	-0.085	-0.006	0.073
20.924	75.000	120.000	100.000	304.000	-2.263	-0.085	-0.006	0.074
29.557	75.000	100.000	100.000	305.000	-2.707	-0.075	-0.008	0.059
29.187	75.000	101.000	100.000	305.000	-2.685	-0.075	-0.008	0.059
28.810	75.000	102.000	100.000	305.000	-2.663	-0.075	-0.008	0.060
28.428	75.000	103.000	100.000	305.000	-2.641	-0.076	-0.008	0.060
28.039	75.000	104.000	100.000	305.000	-2.619	-0.076	-0.008	0.061
27.644	75.000	105.000	100.000	305.000	-2.597	-0.077	-0.007	0.062
27.242	75.000	106.000	100.000	305.000	-2.575	-0.077	-0.007	0.063
26.833	75.000	107.000	100.000	305.000	-2.554	-0.078	-0.007	0.063
26.417	75.000	108.000	100.000	305.000	-2.532	-0.078	-0.007	0.064
25.993	75.000	109.000	100.000	305.000	-2.510	-0.079	-0.007	0.065
25.563	75.000	110.000	100.000	305.000	-2.488	-0.079	-0.007	0.065
25.124	75.000	111.000	100.000	305.000	-2.466	-0.080	-0.007	0.066
24.678	75.000	112.000	100.000	305.000	-2.444	-0.080	-0.007	0.067
24.224	75.000	113.000	100.000	305.000	-2.422	-0.081	-0.007	0.068
23.761	75.000	114.000	100.000	305.000	-2.401	-0.081	-0.007	0.068
23.290	75.000	115.000	100.000	305.000	-2.379	-0.082	-0.006	0.069
22.811	75.000	116.000	100.000	305.000	-2.357	-0.083	-0.006	0.070
22.322	75.000	117.000	100.000	305.000	-2.335	-0.083	-0.006	0.071
21.824	75.000	118.000	100.000	305.000	-2.313	-0.084	-0.006	0.072
21.316	75.000	119.000	100.000	305.000	-2.291	-0.085	-0.006	0.073
20.799	75.000	120.000	100.000	305.000	-2.269	-0.085	-0.006	0.073
29.470	75.000	100.000	100.000	306.000	-2.715	-0.074	-0.008	0.058
29.098	75.000	101.000	100.000	306.000	-2.693	-0.075	-0.008	0.059
28.720	75.000	102.000	100.000	306.000	-2.671	-0.075	-0.008	0.060
28.336	75.000	103.000	100.000	306.000	-2.649	-0.076	-0.008	0.060
27.946	75.000	104.000	100.000	306.000	-2.627	-0.076	-0.008	0.061
27.549	75.000	109.000	100.000	306.000	-2.605	-0.077	-0.007	0.062

27.145	75.000	106.000	100.000	306.000	-2.583	-0.077	-0.007	0.062
26.735	75.000	107.000	100.000	306.000	-2.561	-0.078	-0.007	0.063
26.317	75.000	108.000	100.000	306.000	-2.539	-0.078	-0.007	0.064
25.892	75.000	109.000	100.000	306.000	-2.517	-0.079	-0.007	0.064
25.460	75.000	110.000	100.000	306.000	-2.495	-0.079	-0.007	0.065
25.019	75.000	111.000	100.000	306.000	-2.473	-0.080	-0.007	0.066
24.571	75.000	112.000	100.000	306.000	-2.451	-0.080	-0.007	0.067
24.115	75.000	113.000	100.000	306.000	-2.429	-0.081	-0.007	0.067
23.651	75.000	114.000	100.000	306.000	-2.407	-0.081	-0.007	0.068
23.178	75.000	115.000	100.000	306.000	-2.385	-0.082	-0.006	0.069
22.696	75.000	116.000	100.000	306.000	-2.363	-0.082	-0.006	0.070
22.205	75.000	117.000	100.000	306.000	-2.341	-0.083	-0.006	0.071
21.705	75.000	118.000	100.000	306.000	-2.319	-0.084	-0.006	0.072
21.195	75.000	119.000	100.000	306.000	-2.297	-0.084	-0.006	0.072
20.675	75.000	120.000	100.000	306.000	-2.275	-0.085	-0.006	0.073
29.383	75.000	100.000	100.000	307.000	-2.723	-0.074	-0.008	0.058
29.010	75.000	101.000	100.000	307.000	-2.701	-0.075	-0.008	0.059
28.631	75.000	102.000	100.000	307.000	-2.679	-0.075	-0.008	0.059
28.245	75.000	103.000	100.000	307.000	-2.656	-0.075	-0.008	0.060
27.854	75.000	104.000	100.000	307.000	-2.634	-0.076	-0.008	0.061
27.455	75.000	105.000	100.000	307.000	-2.612	-0.076	-0.007	0.061
27.050	75.000	106.000	100.000	307.000	-2.590	-0.077	-0.007	0.062
26.638	75.000	107.000	100.000	307.000	-2.568	-0.077	-0.007	0.063
26.218	75.000	108.000	100.000	307.000	-2.546	-0.078	-0.007	0.064
25.791	75.000	109.000	100.000	307.000	-2.524	-0.078	-0.007	0.064
25.357	75.000	110.000	100.000	307.000	-2.502	-0.079	-0.007	0.065
24.915	75.000	111.000	100.000	307.000	-2.480	-0.079	-0.007	0.066
24.465	75.000	112.000	100.000	307.000	-2.458	-0.080	-0.007	0.067
24.007	75.000	113.000	100.000	307.000	-2.436	-0.081	-0.007	0.067
23.540	75.000	114.000	100.000	307.000	-2.414	-0.081	-0.007	0.068
23.065	75.000	115.000	100.000	307.000	-2.391	-0.082	-0.006	0.069
22.581	75.000	116.000	100.000	307.000	-2.369	-0.082	-0.006	0.070
22.088	75.000	117.000	100.000	307.000	-2.347	-0.083	-0.006	0.070
21.586	75.000	118.000	100.000	307.000	-2.325	-0.083	-0.006	0.071
21.074	75.000	119.000	100.000	307.000	-2.303	-0.084	-0.006	0.072
20.552	75.000	120.000	100.000	307.000	-2.281	-0.085	-0.006	0.073
29.297	75.000	100.000	100.000	308.000	-2.731	-0.074	-0.008	0.058
28.922	75.000	101.000	100.000	308.000	-2.708	-0.074	-0.008	0.059
28.542	75.000	102.000	100.000	308.000	-2.686	-0.075	-0.008	0.059
28.155	75.000	103.000	100.000	308.000	-2.664	-0.075	-0.008	0.060
27.762	75.000	104.000	100.000	308.000	-2.642	-0.076	-0.008	0.061
27.362	75.000	105.000	100.000	308.000	-2.620	-0.076	-0.007	0.061
26.955	75.000	106.000	100.000	308.000	-2.598	-0.077	-0.007	0.062
26.541	75.000	107.000	100.000	308.000	-2.575	-0.077	-0.007	0.063
26.120	75.000	108.000	100.000	308.000	-2.553	-0.078	-0.007	0.063
25.691	75.000	109.000	100.000	308.000	-2.531	-0.078	-0.007	0.064
25.255	75.000	110.000	100.000	308.000	-2.509	-0.079	-0.007	0.065
24.811	75.000	111.000	100.000	308.000	-2.487	-0.079	-0.007	0.066
24.359	75.000	112.000	100.000	308.000	-2.464	-0.080	-0.007	0.066
23.899	75.000	113.000	100.000	308.000	-2.442	-0.080	-0.007	0.067
23.431	75.000	114.000	100.000	308.000	-2.420	-0.081	-0.007	0.068
22.954	75.000	115.000	100.000	308.000	-2.398	-0.081	-0.006	0.069
22.468	75.000	116.000	100.000	308.000	-2.376	-0.082	-0.006	0.069
21.972	75.000	117.000	100.000	308.000	-2.353	-0.083	-0.006	0.070
21.468	75.000	118.000	100.000	308.000	-2.331	-0.083	-0.006	0.071
20.953	75.000	119.000	100.000	308.000	-2.309	-0.084	-0.006	0.072
20.429	75.000	120.000	100.000	308.000	-2.287	-0.085	-0.006	0.073
29.211	75.000	100.000	100.000	309.000	-2.739	-0.074	-0.008	0.058
28.836	75.000	101.000	100.000	309.000	-2.716	-0.074	-0.008	0.058
28.454	75.000	102.000	100.000	309.000	-2.694	-0.075	-0.008	0.059

28.065	75.000	103.000	100.000	309.000	-2.672	-0.075	-0.008	0.060
27.670	75.000	104.000	100.000	309.000	-2.649	-0.076	-0.008	0.060
27.269	75.000	105.000	100.000	309.000	-2.627	-0.076	-0.007	0.061
26.860	75.000	106.000	100.000	309.000	-2.605	-0.076	-0.007	0.062
26.445	75.000	107.000	100.000	309.000	-2.583	-0.077	-0.007	0.062
26.022	75.000	108.000	100.000	309.000	-2.560	-0.077	-0.007	0.063
25.592	75.000	109.000	100.000	309.000	-2.538	-0.078	-0.007	0.064
25.154	75.000	110.000	100.000	309.000	-2.516	-0.078	-0.007	0.065
24.708	75.000	111.000	100.000	309.000	-2.493	-0.079	-0.007	0.065
24.254	75.000	112.000	100.000	309.000	-2.471	-0.080	-0.007	0.066
23.792	75.000	113.000	100.000	309.000	-2.449	-0.080	-0.007	0.067
23.322	75.000	114.000	100.000	309.000	-2.427	-0.081	-0.007	0.068
22.843	75.000	115.000	100.000	309.000	-2.404	-0.081	-0.006	0.068
22.355	75.000	116.000	100.000	309.000	-2.382	-0.082	-0.006	0.069
21.857	75.000	117.000	100.000	309.000	-2.360	-0.082	-0.006	0.070
21.350	75.000	118.000	100.000	309.000	-2.337	-0.083	-0.006	0.071
20.834	75.000	119.000	100.000	309.000	-2.315	-0.084	-0.006	0.072
20.307	75.000	120.000	100.000	309.000	-2.293	-0.084	-0.006	0.073
29.126	75.000	100.000	100.000	310.000	-2.747	-0.074	-0.008	0.058
28.749	75.000	101.000	100.000	310.000	-2.724	-0.074	-0.008	0.058
28.366	75.000	102.000	100.000	310.000	-2.702	-0.074	-0.008	0.059
27.976	75.000	103.000	100.000	310.000	-2.679	-0.075	-0.008	0.059
27.579	75.000	104.000	100.000	310.000	-2.657	-0.075	-0.008	0.060
27.176	75.000	105.000	100.000	310.000	-2.635	-0.076	-0.007	0.061
26.766	75.000	106.000	100.000	310.000	-2.612	-0.076	-0.007	0.062
26.349	75.000	107.000	100.000	310.000	-2.590	-0.077	-0.007	0.062
25.924	75.000	108.000	100.000	310.000	-2.567	-0.077	-0.007	0.063
25.492	75.000	109.000	100.000	310.000	-2.545	-0.078	-0.007	0.064
25.053	75.000	110.000	100.000	310.000	-2.523	-0.078	-0.007	0.064
24.605	75.000	111.000	100.000	310.000	-2.500	-0.079	-0.007	0.065
24.150	75.000	112.000	100.000	310.000	-2.478	-0.079	-0.007	0.066
23.686	75.000	113.000	100.000	310.000	-2.455	-0.080	-0.007	0.067
23.214	75.000	114.000	100.000	310.000	-2.433	-0.080	-0.007	0.067
22.732	75.000	115.000	100.000	310.000	-2.411	-0.081	-0.006	0.068
22.242	75.000	116.000	100.000	310.000	-2.388	-0.082	-0.006	0.069
21.743	75.000	117.000	100.000	310.000	-2.366	-0.082	-0.006	0.070
21.234	75.000	118.000	100.000	310.000	-2.343	-0.083	-0.006	0.071
20.715	75.000	119.000	100.000	310.000	-2.321	-0.084	-0.006	0.072
20.186	75.000	120.000	100.000	310.000	-2.299	-0.084	-0.006	0.072
29.042	75.000	100.000	100.000	311.000	-2.755	-0.073	-0.008	0.057
28.663	75.000	101.000	100.000	311.000	-2.732	-0.074	-0.008	0.058
28.278	75.000	102.000	100.000	311.000	-2.710	-0.074	-0.008	0.059
27.887	75.000	103.000	100.000	311.000	-2.687	-0.075	-0.008	0.059
27.489	75.000	104.000	100.000	311.000	-2.665	-0.075	-0.008	0.060
27.084	75.000	105.000	100.000	311.000	-2.642	-0.076	-0.007	0.061
26.673	75.000	106.000	100.000	311.000	-2.620	-0.076	-0.007	0.061
26.254	75.000	107.000	100.000	311.000	-2.597	-0.077	-0.007	0.062
25.828	75.000	108.000	100.000	311.000	-2.575	-0.077	-0.007	0.063
25.394	75.000	109.000	100.000	311.000	-2.552	-0.078	-0.007	0.063
24.953	75.000	110.000	100.000	311.000	-2.530	-0.078	-0.007	0.064
24.503	75.000	111.000	100.000	311.000	-2.507	-0.079	-0.007	0.065
24.046	75.000	112.000	100.000	311.000	-2.485	-0.079	-0.007	0.066
23.580	75.000	113.000	100.000	311.000	-2.462	-0.080	-0.007	0.066
23.106	75.000	114.000	100.000	311.000	-2.440	-0.080	-0.007	0.067
22.622	75.000	115.000	100.000	311.000	-2.417	-0.081	-0.006	0.068
22.130	75.000	116.000	100.000	311.000	-2.395	-0.081	-0.006	0.069
21.629	75.000	117.000	100.000	311.000	-2.372	-0.082	-0.006	0.070
21.117	75.000	118.000	100.000	311.000	-2.350	-0.083	-0.006	0.071
20.596	75.000	119.000	100.000	311.000	-2.327	-0.083	-0.006	0.071
20.065	75.000	120.000	100.000	311.000	-2.305	-0.084	-0.006	0.072

28.958	75.000	100.000	100.000	312.000	-2.763	-0.073	-0.008	0.057
28.578	75.000	101.000	100.000	312.000	-2.740	-0.074	-0.008	0.058
28.191	75.000	102.000	100.000	312.000	-2.717	-0.074	-0.008	0.058
27.798	75.000	103.000	100.000	312.000	-2.695	-0.074	-0.008	0.059
27.399	75.000	104.000	100.000	312.000	-2.672	-0.075	-0.008	0.060
26.993	75.000	105.000	100.000	312.000	-2.650	-0.075	-0.007	0.060
26.580	75.000	106.000	100.000	312.000	-2.627	-0.076	-0.007	0.061
26.159	75.000	107.000	100.000	312.000	-2.604	-0.076	-0.007	0.062
25.731	75.000	108.000	100.000	312.000	-2.582	-0.077	-0.007	0.063
25.296	75.000	109.000	100.000	312.000	-2.559	-0.077	-0.007	0.063
24.853	75.000	110.000	100.000	312.000	-2.537	-0.078	-0.007	0.064
24.402	75.000	111.000	100.000	312.000	-2.514	-0.078	-0.007	0.065
23.942	75.000	112.000	100.000	312.000	-2.491	-0.079	-0.007	0.066
23.475	75.000	113.000	100.000	312.000	-2.469	-0.080	-0.007	0.066
22.998	75.000	114.000	100.000	312.000	-2.446	-0.080	-0.007	0.067
22.513	75.000	115.000	100.000	312.000	-2.423	-0.081	-0.006	0.068
22.019	75.000	116.000	100.000	312.000	-2.401	-0.081	-0.006	0.069
21.515	75.000	117.000	100.000	312.000	-2.378	-0.082	-0.006	0.069
21.002	75.000	118.000	100.000	312.000	-2.356	-0.082	-0.006	0.070
20.478	75.000	119.000	100.000	312.000	-2.333	-0.083	-0.006	0.071
19.945	75.000	120.000	100.000	312.000	-2.310	-0.084	-0.006	0.072
28.874	75.000	100.000	100.000	313.000	-2.771	-0.073	-0.008	0.057
28.493	75.000	101.000	100.000	313.000	-2.748	-0.073	-0.008	0.058
28.105	75.000	102.000	100.000	313.000	-2.725	-0.074	-0.008	0.058
27.711	75.000	103.000	100.000	313.000	-2.703	-0.074	-0.008	0.059
27.310	75.000	104.000	100.000	313.000	-2.680	-0.075	-0.008	0.060
26.902	75.000	105.000	100.000	313.000	-2.657	-0.075	-0.007	0.060
26.487	75.000	106.000	100.000	313.000	-2.634	-0.076	-0.007	0.061
26.065	75.000	107.000	100.000	313.000	-2.612	-0.076	-0.007	0.062
25.636	75.000	108.000	100.000	313.000	-2.589	-0.077	-0.007	0.062
25.198	75.000	109.000	100.000	313.000	-2.566	-0.077	-0.007	0.063
24.754	75.000	110.000	100.000	313.000	-2.543	-0.078	-0.007	0.064
24.301	[illegible].000	111.000	100.000	313.000	-2.521	-0.078	-0.007	0.065
23.840	75.000	112.000	100.000	313.000	-2.498	-0.079	-0.007	0.065
23.370	75.000	113.000	100.000	313.000	-2.475	-0.079	-0.007	0.066
22.892	75.000	114.000	100.000	313.000	-2.453	-0.080	-0.007	0.067
22.405	75.000	115.000	100.000	313.000	-2.430	-0.080	-0.006	0.068
21.908	75.000	116.000	100.000	313.000	-2.407	-0.081	-0.006	0.068
21.402	75.000	117.000	100.000	313.000	-2.384	-0.082	-0.006	0.069
20.887	75.000	118.000	100.000	313.000	-2.362	-0.082	-0.006	0.070
20.361	75.000	119.000	100.000	313.000	-2.339	-0.083	-0.006	0.071
19.825	75.000	120.000	100.000	313.000	-2.316	-0.084	-0.006	0.072
28.791	75.000	100.000	100.000	314.000	-2.779	-0.073	-0.008	0.057
28.408	75.000	101.000	100.000	314.000	-2.756	-0.073	-0.008	0.057
28.019	75.000	102.000	100.000	314.000	-2.733	-0.074	-0.008	0.058
27.623	75.000	103.000	100.000	314.000	-2.710	-0.074	-0.008	0.059
27.221	75.000	104.000	100.000	314.000	-2.687	-0.075	-0.008	0.059
26.811	75.000	105.000	100.000	314.000	-2.665	-0.075	-0.007	0.060
26.395	75.000	106.000	100.000	314.000	-2.642	-0.075	-0.007	0.061
25.971	75.000	107.000	100.000	314.000	-2.619	-0.076	-0.007	0.061
25.540	75.000	108.000	100.000	314.000	-2.596	-0.076	-0.007	0.062
25.102	75.000	109.000	100.000	314.000	-2.573	-0.077	-0.007	0.063
24.655	75.000	110.000	100.000	314.000	-2.550	-0.078	-0.007	0.064
24.200	75.000	111.000	100.000	314.000	-2.528	-0.078	-0.007	0.064
23.737	75.000	112.000	100.000	314.000	-2.505	-0.079	-0.007	0.065
23.266	75.000	113.000	100.000	314.000	-2.482	-0.079	-0.007	0.066
22.786	75.000	114.000	100.000	314.000	-2.459	-0.080	-0.007	0.067
22.296	75.000	115.000	100.000	314.000	-2.436	-0.080	-0.006	0.067
21.798	75.000	116.000	100.000	314.000	-2.413	-0.081	-0.006	0.068
21.290	75.000	117.000	100.000	314.000	-2.391	-0.081	-0.006	0.069

20.772	75.000	118.000	100.000	314.000	-2.368	-0.082	-0.006	0.070
20.244	75.000	119.000	100.000	314.000	-2.345	-0.083	-0.006	0.071
19.706	75.000	120.000	100.000	314.000	-2.322	-0.083	-0.006	0.072
28.708	75.000	100.000	100.000	315.000	-2.787	-0.073	-0.008	0.057
28.324	75.000	101.000	100.000	315.000	-2.764	-0.073	-0.008	0.057
27.933	75.000	102.000	100.000	315.000	-2.741	-0.073	-0.008	0.058
27.536	75.000	103.000	100.000	315.000	-2.718	-0.074	-0.008	0.059
27.132	75.000	104.000	100.000	315.000	-2.695	-0.074	-0.008	0.059
26.722	75.000	105.000	100.000	315.000	-2.672	-0.075	-0.007	0.060
26.304	75.000	106.000	100.000	315.000	-2.649	-0.075	-0.007	0.061
25.878	75.000	107.000	100.000	315.000	-2.626	-0.076	-0.007	0.061
25.446	75.000	108.000	100.000	315.000	-2.603	-0.076	-0.007	0.062
25.005	75.000	109.000	100.000	315.000	-2.580	-0.077	-0.007	0.063
24.557	75.000	110.000	100.000	315.000	-2.557	-0.077	-0.007	0.063
24.100	75.000	111.000	100.000	315.000	-2.534	-0.078	-0.007	0.064
23.636	75.000	112.000	100.000	315.000	-2.511	-0.078	-0.007	0.065
23.162	75.000	113.000	100.000	315.000	-2.489	-0.079	-0.007	0.066
22.680	75.000	114.000	100.000	315.000	-2.466	-0.080	-0.007	0.066
22.189	75.000	115.000	100.000	315.000	-2.443	-0.080	-0.006	0.067
21.688	75.000	116.000	100.000	315.000	-2.420	-0.081	-0.006	0.068
21.178	75.000	117.000	100.000	315.000	-2.397	-0.081	-0.006	0.069
20.658	75.000	118.000	100.000	315.000	-2.374	-0.082	-0.006	0.070
20.128	75.000	119.000	100.000	315.000	-2.351	-0.083	-0.006	0.071
19.588	75.000	120.000	100.000	315.000	-2.328	-0.083	-0.006	0.071
28.626	75.000	100.000	100.000	316.000	-2.795	-0.072	-0.008	0.056
28.240	75.000	101.000	100.000	316.000	-2.772	-0.073	-0.008	0.057
27.848	75.000	102.000	100.000	316.000	-2.749	-0.073	-0.008	0.058
27.450	75.000	103.000	100.000	316.000	-2.726	-0.074	-0.008	0.058
27.045	75.000	104.000	100.000	316.000	-2.703	-0.074	-0.008	0.059
26.632	75.000	105.000	100.000	316.000	-2.679	-0.075	-0.007	0.060
26.213	75.000	106.000	100.000	316.000	-2.656	-0.075	-0.007	0.060
25.786	75.000	107.000	100.000	316.000	-2.633	-0.076	-0.007	0.061
25.351	75.000	108.000	100.000	316.000	-2.610	-0.076	-0.007	0.062
24.909	75.000	109.000	100.000	316.000	-2.587	-0.077	-0.007	0.063
24.459	75.000	110.000	100.000	316.000	-2.564	-0.077	-0.007	0.063
24.001	75.000	111.000	100.000	316.000	-2.541	-0.078	-0.007	0.064
23.534	75.000	112.000	100.000	316.000	-2.518	-0.078	-0.007	0.065
23.059	75.000	113.000	100.000	316.000	-2.495	-0.079	-0.007	0.066
22.575	75.000	114.000	100.000	316.000	-2.472	-0.079	-0.007	0.066
22.082	75.000	115.000	100.000	316.000	-2.449	-0.080	-0.006	0.067
21.579	75.000	116.000	100.000	316.000	-2.426	-0.080	-0.006	0.068
21.067	75.000	117.000	100.000	316.000	-2.403	-0.081	-0.006	0.069
20.545	75.000	118.000	100.000	316.000	-2.380	-0.082	-0.006	0.070
20.013	75.000	119.000	100.000	316.000	-2.357	-0.082	-0.006	0.070
19.470	75.000	120.000	100.000	316.000	-2.334	-0.083	-0.006	0.071
28.544	75.000	100.000	100.000	317.000	-2.803	-0.072	-0.008	0.056
28.157	75.000	101.000	100.000	317.000	-2.780	-0.073	-0.008	0.057
27.764	75.000	102.000	100.000	317.000	-2.756	-0.073	-0.008	0.058
27.364	75.000	103.000	100.000	317.000	-2.733	-0.074	-0.008	0.058
26.957	75.000	104.000	100.000	317.000	-2.710	-0.074	-0.008	0.059
26.543	75.000	105.000	100.000	317.000	-2.687	-0.074	-0.007	0.060
26.122	75.000	106.000	100.000	317.000	-2.664	-0.075	-0.007	0.060
25.694	75.000	107.000	100.000	317.000	-2.641	-0.075	-0.007	0.061
25.258	75.000	108.000	100.000	317.000	-2.617	-0.076	-0.007	0.062
24.814	75.000	109.000	100.000	317.000	-2.594	-0.076	-0.007	0.062
24.362	75.000	110.000	100.000	317.000	-2.571	-0.077	-0.007	0.063
23.902	75.000	111.000	100.000	317.000	-2.548	-0.077	-0.007	0.064
23.434	75.000	112.000	100.000	317.000	-2.525	-0.078	-0.007	0.065
22.957	75.000	113.000	100.000	317.000	-2.502	-0.079	-0.007	0.065
22.471	75.000	114.000	100.000	317.000	-2.479	-0.079	-0.007	0.066

21.975	75.000	115.000	100.000	317.000	-2.455	-0.080	-0.006	0.067
21.471	75.000	116.000	100.000	317.000	-2.432	-0.080	-0.006	0.068
20.957	75.000	117.000	100.000	317.000	-2.409	-0.081	-0.006	0.069
20.432	75.000	118.000	100.000	317.000	-2.386	-0.082	-0.006	0.069
19.898	75.000	119.000	100.000	317.000	-2.363	-0.082	-0.006	0.070
19.353	75.000	120.000	100.000	317.000	-2.340	-0.083	-0.006	0.071
28.463	75.000	100.000	100.000	318.000	-2.811	-0.072	-0.008	0.056
28.075	75.000	101.000	100.000	318.000	-2.787	-0.072	-0.008	0.057
27.680	75.000	102.000	100.000	318.000	-2.764	-0.073	-0.008	0.057
27.279	75.000	103.000	100.000	318.000	-2.741	-0.073	-0.008	0.058
26.870	75.000	104.000	100.000	318.000	-2.718	-0.074	-0.008	0.059
26.455	75.000	105.000	100.000	318.000	-2.694	-0.074	-0.007	0.059
26.032	75.000	106.000	100.000	318.000	-2.671	-0.075	-0.007	0.060
25.602	75.000	107.000	100.000	318.000	-2.648	-0.075	-0.007	0.061
25.165	75.000	108.000	100.000	318.000	-2.625	-0.076	-0.007	0.061
24.719	75.000	109.000	100.000	318.000	-2.601	-0.076	-0.007	0.062
24.266	75.000	110.000	100.000	318.000	-2.578	-0.077	-0.007	0.063
23.804	75.000	111.000	100.000	318.000	-2.555	-0.077	-0.007	0.064
23.334	75.000	112.000	100.000	318.000	-2.532	-0.078	-0.007	0.064
22.855	75.000	113.000	100.000	318.000	-2.508	-0.078	-0.007	0.065
22.367	75.000	114.000	100.000	318.000	-2.485	-0.079	-0.007	0.066
21.870	75.000	115.000	100.000	318.000	-2.462	-0.080	-0.006	0.067
21.363	75.000	116.000	100.000	318.000	-2.439	-0.080	-0.006	0.068
20.847	75.000	117.000	100.000	318.000	-2.415	-0.081	-0.006	0.068
20.320	75.000	118.000	100.000	318.000	-2.392	-0.081	-0.006	0.069
19.783	75.000	119.000	100.000	318.000	-2.369	-0.082	-0.006	0.070
19.236	75.000	120.000	100.000	318.000	-2.346	-0.083	-0.006	0.071
28.382	75.000	100.000	100.000	319.000	-2.819	-0.072	-0.008	0.056
27.993	75.000	101.000	100.000	319.000	-2.795	-0.072	-0.008	0.057
27.596	75.000	102.000	100.000	319.000	-2.772	-0.073	-0.008	0.057
27.194	75.000	103.000	100.000	319.000	-2.749	-0.073	-0.008	0.058
26.784	75.000	104.000	100.000	319.000	-2.725	-0.074	-0.008	0.058
26.367	75.000	105.000	100.000	319.000	-2.702	-0.074	-0.007	0.059
25.943	75.000	106.000	100.000	319.000	-2.679	-0.075	-0.007	0.060
25.511	75.000	107.000	100.000	319.000	-2.655	-0.075	-0.007	0.061
25.072	75.000	108.000	100.000	319.000	-2.632	-0.076	-0.007	0.061
24.625	75.000	109.000	100.000	319.000	-2.608	-0.076	-0.007	0.062
24.170	75.000	110.000	100.000	319.000	-2.585	-0.077	-0.007	0.063
23.706	75.000	111.000	100.000	319.000	-2.562	-0.077	-0.007	0.063
23.234	75.000	112.000	100.000	319.000	-2.538	-0.078	-0.007	0.064
22.753	75.000	113.000	100.000	319.000	-2.515	-0.078	-0.007	0.065
22.263	75.000	114.000	100.000	319.000	-2.492	-0.079	-0.007	0.066
21.764	75.000	115.000	100.000	319.000	-2.468	-0.079	-0.006	0.067
21.256	75.000	116.000	100.000	319.000	-2.445	-0.080	-0.006	0.067
20.737	75.000	117.000	100.000	319.000	-2.422	-0.081	-0.006	0.068
20.209	75.000	118.000	100.000	319.000	-2.398	-0.081	-0.006	0.069
19.670	75.000	119.000	100.000	319.000	-2.375	-0.082	-0.006	0.070
19.120	75.000	120.000	100.000	319.000	-2.351	-0.082	-0.006	0.071
28.302	75.000	100.000	100.000	320.000	-2.827	-0.072	-0.008	0.056
27.911	75.000	101.000	100.000	320.000	-2.803	-0.072	-0.008	0.056
27.513	75.000	102.000	100.000	320.000	-2.780	-0.073	-0.008	0.057
27.109	75.000	103.000	100.000	320.000	-2.756	-0.073	-0.008	0.058
26.698	75.000	104.000	100.000	320.000	-2.733	-0.073	-0.008	0.058
26.280	75.000	105.000	100.000	320.000	-2.709	-0.074	-0.007	0.059
25.854	75.000	106.000	100.000	320.000	-2.686	-0.074	-0.007	0.060
25.421	75.000	107.000	100.000	320.000	-2.662	-0.075	-0.007	0.060
24.980	75.000	108.000	100.000	320.000	-2.639	-0.075	-0.007	0.061
24.531	75.000	109.000	100.000	320.000	-2.615	-0.076	-0.007	0.062
24.074	75.000	110.000	100.000	320.000	-2.592	-0.076	-0.007	0.063
23.609	75.000	111.000	100.000	320.000	-2.569	-0.077	-0.007	0.063

```
    23.135      75.000     112.000     100.000     320.000      -2.545      -0.077      -0.007       0.064
    22.652      75.000     113.000     100.000     320.000      -2.522      -0.078      -0.007       0.065
    22.161      75.000     114.000     100.000     320.000      -2.498      -0.079      -0.007       0.066
    21.659      75.000     115.000     100.000     320.000      -2.475      -0.079      -0.006       0.066
    21.149      75.000     116.000     100.000     320.000      -2.451      -0.080      -0.006       0.067
    20.628      75.000     117.000     100.000     320.000      -2.428      -0.080      -0.006       0.068
    20.098      75.000     118.000     100.000     320.000      -2.404      -0.081      -0.006       0.069
    19.556      75.000     119.000     100.000     320.000      -2.381      -0.082      -0.006       0.070
    19.005      75.000     120.000     100.000     320.000      -2.357      -0.082      -0.006       0.070
$ log
Accounting information:
Buffered I/O count:          955     Peak WS size:              1032
Direct I/O count:            241     Peak pagefile size:        4454
Page faults:                2496     Mounted volumes:              0
Charged CPU time   0 00:00:06.78     Elapsed time:      0 00:06:33.08

Charge information:
Buffered I/O:     $     0.00000NC    Peak WS:          $     0.00000NC
Direct I/O:               0.02651    Peak pagefile:            0.00000NC
Page faults:              0.00000NC  Mounted volumes:          0.00000
Charged CPU:              0.16558    Elapsed time:             0.03821
Total Charges:    $     0.23030
  SKERCEL       logged out at  8-FEB-1990 14:32:21.33
```

APPENDIX J

CODE LISTING: PROTOTYPE I/O PROGRAMS

```
/* UT Measurement and Control Engineering Center                          */
/*                                                                        */
/* Stephen W. Kercel                                        March 2, 1990 */
/*                                                                        */
/* driver.c                                                               */
/*                                                                        */
/* Program to move the Daedal 3-axis positioner and to read its position  */
/* from the encoder.                                                      */
/*                                                                        */
/* This program moves the positioner, prints out its new coordinates, moves*/
/* it back to its point of origin, and prints out those coordinates. The  */
/* purpose of this program is to demonstrate the functions used to move the*/
/* positioner and read the encoder. In the overall control program for the */
/* Raman system, the functions will probably be copied from this program. */
/* This program supersedes the C program in the Daedal MC 5300 Controller */
/* manual.                                                                */
/*                                                                        */
/* CAUTION:                                                               */
/* This program is not portable. It was developed under Microsoft Quick C, */
/* using Microsoft library functions for I/O. The I/O messages are hardware*/
/* dependent, being specifically designed for the Daedal PC23 indexer.    */
/*                                                                        */
/* DO NOT COMPILE THIS PROGRAM WITH ANY COMPILER EXCEPT MICROSOFT QUICK C */
/* VERSION 2.01 OR HIGHER.                                                */
/*                                                                        */
/* RUN THE EXECUTABLE CODE ONLY ON A PC EQUIPPED WITH DAEDAL PC23 CIRCUIT */
/* BOARD. MAKE SURE THE I/O ADDRESS SWITCH SETTINGS ON THE BOARD AGREE WITH*/
/* THE I/O ADDRESS IN THE DECLARED CONSTANTS OF THIS PROGRAM.             */
/*                                                                        */
/* FAILURE TO OBSERVE THE ABOVE PRECAUTIONS WILL LEAD TO UNEXPECTED RESULTS*/
/*                                                                        */
/* Input Data:                                                            */
/*  Positioner command strings - hard coded into main function           */
/*                                                                        */
/* Output Data:                                                           */
/*  New stage location - Stage moves; new coordinates displayed           */
/*                                                                        */
/* Functions called from main():                                          */
/*  initialize() - Assures that PC23 board is ready to accept commands    */
/*  writecmd() - Sends a single character to the PC23 board               */
/*  readanswer() - Reads a single character from the PC23 board           */
/**************************************************************************/

/*headers*/
#include <stdio.h>  /* I/O headers standard for most C programs*/
#include <string.h> /* headers for string handling -- Microsoft library*/
#include <conio.h> /* I/O headers -- Microsoft library*/

/*defined constants*/
#define TRUE 1
#define FALSE 0
#define ADDRESS0 0x300
#define ADDRESS1 0x310
#define MAXINT 32000
#define RETURN 0xD

/* In addition to the general purpose defined constants listed above, */
/* the control of the positioner makes use of various status and control*/
/* bit masks. Pages 22-30 of the PC23-03 manual describe the functions of*/
/* each registers bits (with the exception of the status registers bit 7*/
/* which indicates if the PC23 buffer is over half full). Page 30 gives*/
/* a step by step procedure of the handshaking process. Page 62 explains*/
/* the differences between the binary write (time distance) function and*/
/* the normal write function.                                          */
```

```
#define FAIL 0x20 /* 0010 0000 */
#define START_M 0x7F /* 0111 1111 */
#define INTCLR 0x20 /* 0010 0000 */
#define RESTART 0x17 /* 0001 0111 */
#define POSITIVE 0x8000 /* 1000 0000 0000 0000 */
#define ALDONE 0x7 /* 0111 */
#define STOP 0x64 /* 0110 0100 */
#define START 0x40 /* 0100 0000 */
#define CB 0x60 /* 0110 0000 */
#define CB1 0x61 /* 0110 0001 */
#define IDB_M 0x10 /* 0001 0000 */
#define IDB 0x70 /* 0111 0000 */
#define IDB1 0x71 /* 0111 0001 */
#define ODB 0x8 /* 1000 */
#define ACK 0xE0 /* 1110 0000 */
#define BFULL 0x80 /* 1000 0000 */ /*checks buffer status*/

/* All preceding compiler directives are overhead. The actual program*/
/* starts here. No global variables have been declared.              */

main()
{
        /*local declarations and initializations */

        char message[81] =
        " 1A10 1V10 1D100000 2A10 2V10 2D100000 G12 1PR ";
        /* string that moves positioner */
        char answer[81] = ""; /* holder for string that reports position */
        int address = ADDRESS0;/* address of I/O board for this positioner */
        int goflag = TRUE;/* 1=> OK to procede */

        /*main body of function*/

        /*move positioner and report its new position*/
        initialize(address,&goflag);       /* set up PC23 */
        if (goflag) /*procede only if PC23 got set up ok*/
         {
                writecmd(address,message); /* move positioner */
                readanswer(address,answer); /* find location of positioner */
                printf("\nAxis is now at position=%s\n", answer);

                /*move positioner back to start position*/
                strcpy(message," 1H 2H G12 1PR "); /*update command string*/
                writecmd(address,message); /*move positioner*/
                /*wait for positioner to return to start position*/
                /*wait for motors to stop moving*/
                while ((inp(address+1) & ALDONE) != ALDONE) /* bitwise AND */
                        ;
                readanswer(address,answer);/*find location of positioner*/
                printf("\nAxis is now at position=%s\n",answer);
         }
}

/* function initialize()                                                  */
/*                                                                        */
/* Function to reset the PC23 board                                       */
/* This function assures that the board is ready to accept commands from  */
/* program. The PC23 must have its address set for the first value in the */
/* user data file, or the program will exit. (ie goflag will clear)       */
/*                                                                        */
/* input data:                                                            */
/*  adr - I/O address of port                                             */
/*                                                                        */
/* output data:                                                           */
/*  gofrag - TRUE => port set up ok                                       */
```

```
/*                                                                          */
/* functions called: none                                                   */
/*                                                                          */
/**************************************************************************/

initialize(adr,gofrag)
int adr; /*address of PC23 board*/
int *gofrag;/*TRUE => board set up OK */
{
        /* local declarations and initializations*/
        int i=0;/*loop counter*/

        /*function starts here*/

        /* Page 30 step 1 */
        /* STOP mask forces bits 2, 5 and 6 high*/
        /* bit 2 times out and stops the watchdog timer */
        /* bit 5 restarts the watchdog timer */
        /* bit 6 resets the interrupt latch */
        outp(adr+1,STOP); /* stop whatever the port is doing */

        /* Page 30 step 2 */
        /* wait for bit 5 to go high  indicates port fail*/
        while (!((inp(adr+1)) & FAIL))
                ;

        /* Page 30 step 3 */
        /* START mask forces bit 6 high */
        /* this resets the interrupt latch */
        outp(adr+1,START);

        /* Page 30 step 4 */
        /* CB mask forces bits 5 and 6 high */
        /* this resets interrupt latch and watchdog timer */
        outp(adr+1,CB);

        /* Page 30 step 5 */
        /* START_M mask is all bits high but 7 */
        /* RESTART mask is bits 0, 1, 2 and 4  high */
        /* indicates motors not running and input data buffer ready */
        /* loop causes the system to wait for this pattern */
        /* to appear, but it gives up after 32000 iterations */
        while (((inp(adr+1) & START_M) != RESTART) && (i++ != MAXINT))
                ;

        if (i >= MAXINT) /* PC23 timed out */
         {
                printf("Initialization of PC23 timed out\n");
                *gofrag = FALSE;/* will force main() to quit */
         }
        else
         {
                *gofrag = TRUE; /*all is well - PC23 did initialize*/

                /* Page 30 step 6 */
                /* INTCLR mask is but 5 high */
                /* restarts watchdog timer */
                outp(adr+1,INTCLR);

                /* Page 30 step 7 */
                /* CB mask forces bits 5 and 6 high */
                /* this resets interrupt latch and watchdog timer */
                outp(adr+1,CB);
         }
}
```

```
/* writecmd()                                                              */
/*                                                                         */
/* function writes a command string to the PC23 port                       */
/* program writes command to port by sending one character at a time       */
/*                                                                         */
/* input data:                                                             */
/*  adr - address of port                                                  */
/*  msg - command message                                                  */
/*                                                                         */
/* output data:                                                            */
/*  s - individual message characters sent to port                         */
/*                                                                         */
/* functions called:                                                       */
/*  writech() - send one character to port                                 */
/*                                                                         */
/*************************************************************************/

writecmd(adr,msg)
int adr;/*address of PC23 board*/
char msg[];/*command string*/
{
        /*this loop starts by pointing at the initial character in */
        /*the array. it writes the character to the address then advances*/
        /*the string pointer. when the thing pointed at by the string */
        /*pointer is 0, as at the end of string marker, the loop quits */
        while (*msg) /*while thing pointed at is not zero, keep going*/
                writech(adr,*msg++); /*write character being pointed at*/
}

/* function readanswer()                                                   */
/*                                                                         */
/* function reads status request response string from PC23                 */
/*                                                                         */
/* input data:                                                             */
/*  addr - address supplied by calling function                            */
/*  msg - string received from PC23 port                                   */
/*                                                                         */
/* output data:                                                            */
/*  msg - string passed back to calling function                           */
/*                                                                         */
/* functions called:                                                       */
/*  readch - reads one character from port                                 */
/*                                                                         */
/*************************************************************************/

readanswer(adr,msg)
int adr;/* address of PC23 port */
char msg[];/* holder for string */
{
        /*go get a character from the port*/
        /*write it to the address pointed at by msg*/
        /*advance pointer to next address*/
        /*keep going until you get end of line marker*/
        while ((*msg++ = readch(adr)) != RETURN)
                ;
        *msg = '\0';/*end of string*/
}

/* function writech()                                                      */
/*                                                                         */
/* Function writes a single character to the PC23 port                     */
/*                                                                         */
/* Input data:                                                             */
/*  addr - address of PC23 port                                            */
```

```
/*  alpha - character passed in from calling function                    */
/*                                                                       */
/* Output data:                                                          */
/*  alpha - character written to port                                    */
/*                                                                       */
/* Functions called: None                                                */
/*                                                                       */
/*************************************************************************/

writech(addr,alpha)
int addr;/*address of PC23 port*/
char alpha;/*character being sent*/
/* Page 30 step 1 - ASCII character passed as parameter */

{
        /* Page 30 step 2 */
        /*wait until bit 4 becomes 1*/
        /* wait for input data buffer to become ready to receive data*/
        while(!(inp(addr+1) & IDB_M))
                ;

        /* Page 30 step 3 */
        outp(addr,alpha);/*send character*/

        /* Page 30 step 4 */
        /*raise bits 4 5 6 */
        /* bit 4 transmit character from buffer to system*/
        /* bit 5 reset watchdog timer */
        /* bit 6 reset interrupt latch */
        outp(addr+1,IDB);/*send command byte*/

        /* Page 30 step 5 */
        /*wait for input data buffer to clear*/
        while(inp(addr+1) & IDB_M)
                ;
        /* Page 30 step 6 */
        /*reset watchdog timer*/
        /*reset interrupt latch*/
        outp(addr+1,CB);

        /* This step not described on page 30 */
        /*wait for input data buffer to become ready to receive data*/
        while(!(inp(addr+1) & IDB_M))
                ;
}

/*function readch()                                                      */
/*                                                                       */
/* Function reads one character of a PC23 response to a status request   */
/* returns character response.                                           */
/*                                                                       */
/* Input data:                                                           */
/*  addr - address provided by calling function                          */
/*  alpha - character provided by PC23                                   */
/*                                                                       */
/* Output data:                                                          */
/*  alpha - character returned to calling function                       */
/*                                                                       */
/* Functions called: None                                                */
/*                                                                       */
/*************************************************************************/

readch(addr)
int addr;/*address of PC23 port*/
{
```

```
        /* Page 30 step 1 */
        char alpha=0;/*character to be fetched from PC23 and returned*/

        /* Page 30 step 2 */
        /*wait for a 1 at bit 3 */
        /*wait until output data buffer is ready*/
        while(!(inp(addr+1) & 0DB))
                ;

        /* Page 30 step 3 */
        alpha = inp(addr);/*fetch a character from output data buffer*/

        /* Page 30 step 4 */
        /* bit 7 I got it */
        /* bit 6 reset interrupt latch */
        /* bit 5 reset watchdog timer */
        outp(addr+1,ACK);/*acknowledge receipt of character*/

        /* Page 30 step 5 */
        /*wait until output data buffer is not ready */
        while(inp(addr+1) & 0DB)
                ;

        /* Page 30 step 6 */
        /*reset interrupt latch and watchdog timer*/
        outp(addr+1,CB);
        return(alpha);
}
```

```
/* UT Measurement and Control Engineering Center                          */
/*                                                                        */
/* Stephen W. Kercel                                        April 18, 1990 */
/*                                                                        */
/* runxkeya.c                                                             */
/*                                                                        */
/*                                                                        */
/* Program gets command strings interactively from the user, and issues   */
/* them to the positioner at the specified address to                     */
/* move the Daedal 3-axis positioner and read its position                */
/* from the encoder.                                                      */
/*                                                                        */
/* CAUTION:                                                               */
/* This program is not portable. It was developed under Microsoft Quick C, */
/* using Microsoft library functions for I/O. The I/O messages are hardware*/
/* dependent, being specifically designed for the Daedal PC23 indexer.    */
/*                                                                        */
/* DO NOT COMPILE THIS PROGRAM WITH ANY COMPILER EXCEPT MICROSOFT QUICK C  */
/* VERSION 2.01 OR HIGHER.                                                */
/*                                                                        */
/* RUN THE EXECUTABLE CODE ONLY ON A PC EQUIPPED WITH DAEDAL PC23 CIRCUIT  */
/* BOARD. MAKE SURE THE I/O ADDRESS SWITCH SETTINGS ON THE BOARD AGREE WITH*/
/* THE I/O ADDRESS IN THE DECLARED CONSTANTS OF THIS PROGRAM.             */
/*                                                                        */
/* FAILURE TO OBSERVE THE ABOVE PRECAUTIONS WILL LEAD TO UNEXPECTED RESULTS*/
/*                                                                        */
/* Input Data:                                                            */
/*  positioner command strings - from keyboard interactively             */
/*  positioner - specified by user 1 or 0                                 */
/*                                                                        */
/* Output Data:                                                           */
/*  New stage location - Stage moves                                      */
/*                                                                        */
/* Functions called from main():                                          */
/*  inp_data() - querys user for positioner                               */
/*  read_a_line() - gets a command string from user                       */
/*  initialize() - Assures that PC23 board is ready to accept commands    */
/*  writecmd() - Sends a single character to the PC23 board               */
/*  readanswer() - Reads a single character from the PC23 board           */
/**************************************************************************/

/*headers*/
#include <stdio.h>  /* I/O headers standard for most C programs*/
#include <string.h> /* headers for string handling -- Microsoft library*/
#include <conio.h> /* I/O headers -- Microsoft library*/
#include <stdlib.h> /* Microsoft extensions */

/*defined constants*/
#define TRUE 1
#define FALSE 0
#define ADDRESS0 0x300
#define ADDRESS1 0x310
#define MAXINT 32000
#define RETURN 0xD

/* In addition to the general purpose defined constants listed above, */
/* the control of the positioner makes use of various status and control*/
/* bit masks. Pages 22-30 of the PC23-03 manual describe the functions of*/
/* each registers bits (with the exception of the status registers bit 7*/
/* which indicates if the PC23 buffer is over half full). Page 30 gives*/
/* a step by step procedure of the handshaking process. Page 62 explains*/
/* the differences between the binary write (time distance) function and*/
/* the normal write function.                                          */

#define FAIL 0x20 /* 0010 0000 */
```

```
#define START_M 0x7F /* 0111 1111 */
#define INTCLR 0x20 /* 0010 0000 */
#define RESTART 0x17 /* 0001 0111 */
#define POSITIVE 0x8000 /* 1000 0000 0000 0000 */
#define ALDONE 0x7 /* 0111 */
#define STOP 0x64 /* 0110 0100 */
#define START 0x40 /* 0100 0000 */
#define CB 0x60 /* 0110 0000 */
#define CB1 0x61 /* 0110 0001 */
#define IDB_M 0x10 /* 0001 0000 */
#define IDB 0x70 /* 0111 0000 */
#define IDB1 0x71 /* 0111 0001 */
#define ODB 0x8 /* 1000 */
#define ACK 0xE0 /* 1110 0000 */
#define BFULL 0x80 /* 1000 0000 */ /*checks buffer status*/

/* All preceding compiler directives are overhead. The actual program*/
/* starts here. No global variables have been declared.              */

main()
{
        int readflag;/* TRUE => read a message from PC23 */
        int address;/*address of positioner*/
        int goflag = TRUE;/*1 => positioner initialized */
        int quitflag = FALSE;/* TRUE => halt return to DOS */
        char s[80];/*file input buffer*/

        inp_data(&address);/*query user*/
        initialize(address,&goflag);/*set up PC23*/
        if (goflag)
         {
                /*if positioner is there, write to it */
                while(!quitflag)
                 {
                        read_a_line(s,&readflag,&quitflag);
                        if (!quitflag)
                         {
                                printf("\nBefore write -- string is %s",s);
                                writecmd(address,s);
                                printf("\nAfter write");
                                if (readflag)
                                 {
                                  readanswer(address,s);
                                  printf("\n Response is %s \n",s);
                                 }
                         }
                 }
         }
        else
                        printf("\nPositioner did not initialize \n");

}

/* function read_a_line()                                                */
/*                                                                       */
/* gets a positioner command line from the user                          */
/*                                                                       */
/* input data:                                                           */
/*  s - string furnished by user                                         */
/*                                                                       */
/* output data (returned as parameters):                                 */
/*  s - command string                                                   */
/*  rdfrag - TRUE => read a message from the positioner                  */
/*  quitfrag - TRUE => return to DOS                                     */
/*                                                                       */
```

```
/* functions called: none                                                    */
/****************************************************************************/

read_a_line(s,rdfrag,quitfrag)
char s[];/*input data line*/
int *rdfrag;/*TRUE => read message from positioner*/
int *quitfrag;/*TRUE => return to DOS*/

{
        /*local declarations*/
        char loc_buf[81];/*local key buffer*/
        char status[81];/*status message string*/
        char stat_flag;/*first letter of status message*/
        char go_flag;/*first letter of command message*/

        /*function starts here*/
        printf("\n");
        printf("\Next positioner command (Ctrl-X to Quit): ");
        fflush(stdin);
        gets(loc_buf);
        strcpy(s,loc_buf);
        go_flag = *s;
        strcat(s," ");
        if (go_flag == ('X'-0X40))
                *quitfrag = TRUE;
        else
         {
                *quitfrag = FALSE;
                printf("\nRead a status message (Y or N)? ");
                scanf("%s",loc_buf);
                strcpy(status,loc_buf);
                stat_flag = *status;
                if ((stat_flag == 'Y') || (stat_flag == 'y'))
                        *rdfrag = TRUE;
                else
                        *rdfrag = FALSE;
         }
}

/* function inp_data()                                                      */
/*                                                                          */
/* gets filename and positioner from user                                   */
/*                                                                          */
/* input data (interactive from user):                                      */
/*  file_name - file containing commands                                    */
/*  positioner - number of positioner 1 or 0                                */
/*                                                                          */
/* output data (passed to calling function):                                */
/*  file_name - file containing commands                                    */
/*  addr - I/O address of positioner                                        */
/*                                                                          */
/* functions called: none                                                   */
/*                                                                          */
/****************************************************************************/

inp_data(addr)
int *addr;/* address of board*/
{
        int positioner = -1;/*positioner number*/
        char s[81];/*local key buffer*/

        while ((positioner != 0) && (positioner !=1))
         {
                printf("\nPositioner number (0 or 1): ");
                scanf("%s",s);
```

```
                positioner = atoi(s);
        }
        if (positioner == 0)
                *addr = ADDRESS0;
        else
                *addr = ADDRESS1;
}

/* function initialize()                                                    */
/*                                                                          */
/* Function to reset the PC23 board                                         */
/* This function assures that the board is ready to accept commands from    */
/* program. The PC23 must have its address set for the first value in the   */
/* user data file, or the program will exit. (ie goflag will clear)         */
/*                                                                          */
/* input data:                                                              */
/*  adr - I/O address of port                                               */
/*                                                                          */
/* output data:                                                             */
/*  gofrag - TRUE => port set up ok                                         */
/*                                                                          */
/* functions called: none                                                   */
/*                                                                          */
/****************************************************************************/

initialize(adr,gofrag)
int adr; /*address of PC23 board*/
int *gofrag;/*TRUE => board set up OK */
{
        /* local declarations and initializations*/
        int i=0;/*loop counter*/

        /*function starts here*/

        /* Page 30 step 1 */
        /* STOP mask forces bits 2, 5 and 6 high*/
        /* bit 2 times out and stops the watchdog timer */
        /* bit 5 restarts the watchdog timer */
        /* bit 6 resets the interrupt latch */
        outp(adr+1,STOP); /* stop whatever the port is doing */

        /* Page 30 step 2 */
        /* wait for bit 5 to go high  indicates port fail*/
        while (!((inp(adr+1)) & FAIL))
                ;

        /* Page 30 step 3 */
        /* START mask forces bit 6 high */
        /* this resets the interrupt latch */
        outp(adr+1,START);

        /* Page 30 step 4 */
        /* CB mask forces bits 5 and 6 high */
        /* this resets interrupt latch and watchdog timer */
        outp(adr+1,CB);

        /* Page 30 step 5 */
        /* START_M mask is all bits high but 7 */
        /* RESTART mask is bits 0, 1, 2 and 4  high */
        /* indicates motors not running and input data buffer ready */
        /* loop causes the system to wait for this pattern */
        /* to appear, but it gives up after 32000 iterations */
        while (((inp(adr+1) & START_M) != RESTART) && (i++ != MAXINT))
                ;
```

```
        if (i >= MAXINT) /* PC23 timed out */
         {
                printf("Initialization of PC23 timed out\n");
                *gofrag = FALSE;/* will force main() to quit */
         }
        else
         {
                *gofrag = TRUE; /*all is well - PC23 did initialize*/

                /* Page 30 step 6 */
                /* INTCLR mask is but 5 high */
                /* restarts watchdog timer */
                outp(adr+1,INTCLR);

                /* Page 30 step 7 */
                /* CB mask forces bits 5 and 6 high */
                /* this resets interrupt latch and watchdog timer */
                outp(adr+1,CB);
         }
}

/* writecmd()                                                              */
/*                                                                         */
/* function writes a command string to the PC23 port                       */
/* program writes command to port by sending one character at a time       */
/*                                                                         */
/* input data:                                                             */
/*  adr - address of port                                                  */
/*  msg - command message                                                  */
/*                                                                         */
/* output data:                                                            */
/*  s - individual message characters sent to port                         */
/*                                                                         */
/* functions called:                                                       */
/*  writech() - send one character to port                                 */
/*                                                                         */
/*************************************************************************/

writecmd(adr,msg)
int adr;/*address of PC23 board*/
char msg[];/*command string*/
{
        /*this loop starts by pointing at the initial character in */
        /*the array. it writes the character to the address then advances*/
        /*the string pointer. when the thing pointed at by the string */
        /*pointer is 0, as at the end of string marker, the loop quits */
        while (*msg) /*while thing pointed at is not zero, keep going*/
                writech(adr,*msg++); /*write character being pointed at*/
}

/* function readanswer()                                                   */
/*                                                                         */
/* function reads status request response string from PC23                 */
/*                                                                         */
/* input data:                                                             */
/*  addr - address supplied by calling function                            */
/*  msg - string received from PC23 port                                   */
/*                                                                         */
/* output data:                                                            */
/*  msg - string passed back to calling function                           */
/*                                                                         */
/* functions called:                                                       */
/*  readch - reads one character from port                                 */
/*                                                                         */
/*************************************************************************/
```

```
readanswer(adr,msg)
int adr;/* address of PC23 port */
char msg[];/* holder for string */
{
        /*go get a character from the port*/
        /*write it to the address pointed at by msg*/
        /*advance pointer to next address*/
        /*keep going until you get end of line marker*/
        while ((*msg++ = readch(adr)) != RETURN)
                ;
        *msg = '\0';/*end of string*/
}

/* function writech()                                                    */
/*                                                                       */
/* Function writes a single character to the PC23 port                   */
/*                                                                       */
/* Input data:                                                           */
/*  addr - address of PC23 port                                          */
/*  alpha - character passed in from calling function                    */
/*                                                                       */
/* Output data:                                                          */
/*  alpha - character written to port                                    */
/*                                                                       */
/* Functions called: None                                                */
/*                                                                       */
/*************************************************************************/

writech(addr,alpha)
int addr;/*address of PC23 port*/
char alpha;/*character being sent*/
/* Page 30 step 1 - ASCII character passed as parameter */

{
        /* Page 30 step 2 */
        /*wait until bit 4 becomes 1*/
        /* wait for input data buffer to become ready to receive data*/
        while(!(inp(addr+1) & IDB_M))
                ;

        /* Page 30 step 3 */
        outp(addr,alpha);/*send character*/

        /* Page 30 step 4 */
        /*raise bits 4 5 6 */
        /* bit 4 transmit character from buffer to system*/
        /* bit 5 reset watchdog timer */
        /* bit 6 reset interrupt latch */
        outp(addr+1,IDB);/*send command byte*/

        /* Page 30 step 5 */
        /*wait for input data buffer to clear*/
        while(inp(addr+1) & IDB_M)
                ;
        /* Page 30 step 6 */
        /*reset watchdog timer*/
        /*reset interrupt latch*/
        outp(addr+1,CB);

        /* This step not described on page 30 */
        /*wait for input data buffer to become ready to receive data*/
        while(!(inp(addr+1) & IDB_M))
                ;
}
```

```
/*function readch()                                                          */
/*                                                                          */
/* Function reads one character of a PC23 response to a status request     */
/* returns character response.                                              */
/*                                                                          */
/* Input data:                                                              */
/*  addr - address provided by calling function                             */
/*  alpha - character provided by PC23                                      */
/*                                                                          */
/* Output data:                                                             */
/*  alpha - character returned to calling function                          */
/*                                                                          */
/* Functions called: None                                                   */
/*                                                                          */
/****************************************************************************/

readch(addr)
int addr;/*address of PC23 port*/
{
        /* Page 30 step 1 */
        char alpha=0;/*character to be fetched from PC23 and returned*/

        /* Page 30 step 2 */
        /*wait for a 1 at bit 3 */
        /*wait until output data buffer is ready*/
        while(!(inp(addr+1) & 0DB))
                ;

        /* Page 30 step 3 */
        alpha = inp(addr);/*fetch a character from output data buffer*/

        /* Page 30 step 4 */
        /* bit 7 I got it */
        /* bit 6 reset interrupt latch */
        /* bit 5 reset watchdog timer */
        outp(addr+1,ACK);/*acknowledge receipt of character*/

        /* Page 30 step 5 */
        /*wait until output data buffer is not ready */
        while(inp(addr+1) & 0DB)
                ;

        /* Page 30 step 6 */
        /*reset interrupt latch and watchdog timer*/
        outp(addr+1,CB);
        return(alpha);
}
```

APPENDIX K

CODE LISTING: INSTRUMENT CONTROL PROGRAM

```
/* UT Measurement and Control Engineering Center                          */
/*                                                                        */
/* Stephen W. Kercel                                        June 14, 1990 */
/*                                                                        */
/* cycler4.c                                                              */
/*                                                                        */
/* Program polls the hardware and files. If not ready, it returns to DOS.  */
/* If files and hardware are ready, program cycles through each of 5 sample*/
/* cells. Each cycle consists of blocking the laser beam with the beam     */
/* dump, positioning the excitation positioner to the proper coordinates,  */
/* unblocking the laser beam, positioning the receiving positioner to the  */
/* proper coordinates, collecting a spectrum of the sample from the        */
/* spectrometer, calculating the composition vector from the spectrum      */
/* vector and calibration matrix, and writing the composition message to   */
/* the output data file. At the end of the cycle, the program polls the key*/
/* buffer to see if a ctrl-X has been issued. If not, the cycle repeats for*/
/* the next sample cell. When the user issues a ctrl-X, the program does an*/
/* orderly exit, resetting the spectrometer and closing the output file.   */
/*                                                                        */
/* CAUTION:                                                               */
/* This program is not portable. It was developed under Microsoft Quick C, */
/* using Microsoft and National Instrument library functions.             */
/* The I/O messages are hardware dependent,                               */
/* being specifically designed for two Daedal PC23 indexers, a            */
/* Perkin-Elmer 1700 interferometer connected via a National Instruments  */
/* IEEE-488 board, and a TTL controlled solenoid run via the printer port. */
/*                                                                        */
/* DO NOT COMPILE THIS PROGRAM WITH ANY COMPILER EXCEPT MICROSOFT QUICK C  */
/* VERSION 2.01 OR HIGHER.                                                */
/*                                                                        */
/* RUN THE EXECUTABLE CODE ONLY ON A PC EQUIPPED AS DESCRIBED ABOVE.      */
/* MAKE SURE THE I/O ADDRESS SWITCH SETTINGS ON THE BOARDS AGREE WITH     */
/* THE I/O ADDRESSES IN THE DECLARED CONSTANTS OF THIS PROGRAM.           */
/*                                                                        */
/* FAILURE TO OBSERVE THE ABOVE PRECAUTIONS WILL LEAD TO UNEXPECTED RESULTS*/
/*                                                                        */
/* Input Data:                                                            */
/*  Files created by user with text editor:                               */
/*   cella0.dat through cella4.dat - contain data for each cell           */
/*  Files created by chemometric training program:                        */
/*   matrix0.dat through matrix4.dat - contain calibration matrix         */
/*                                     for each cell                      */
/*  Vector obtained from spectrometer:                                    */
/*   spectrum0 - spectrum of current sample                               */
/*                                                                        */
/* Output Data (sent to messages.dat):                                    */
/*  Composition message string -- one string per cycle                    */
/*                                                                        */
/* Functions called from main():                                          */
/*  pos_there() - Assures that PC23 boards are ready to accept commands   */
/*  set_mode() - sets PC23s to absolute positioning mode                  */
/*  dofiles() - initializes arrays from cell and matrix files             */
/*  gig1700() - assures that the spectrometer is there                    */
/*  flip_dump() - moves beam dump solenoid                                */
/*  move_stage() - move stage to next set of absolute coordinates         */
/*  get_spectrum() - collect spectrum through spectrometer                */
/*  composition() - compute composition vector                            */
/*  erbyte() - compute the error byte                                     */
/*  write_log() - send composition message to log file                    */
/*  ungig1700() - return spectrometer to conventional mode                */
/*  read_a_line() - reads key buffer checks for quit signal               */
/************************************************************************/

/*headers*/
#include <stdio.h>  /* I/O headers standard for most C programs*/
```

```
#include <string.h> /* headers for string handling -- Microsoft library*/
#include <conio.h> /* I/O headers -- Microsoft library*/
#include <stdlib.h> /* Microsoft extensions */
#include <time.h> /* Microsoft extensions */

/*defined constants*/
#define TRUE 1
#define FALSE 0
#define ADDRESS0 0x300 /*positioner 0 address*/
#define ADDRESS1 0x310 /*positioner 1 address*/
#define ADDRESS2 0x378 /*beam dump address*/
#define TIMEOUT 1200 /*maximum spectrum scan time in seconds*/
#define MAXINT 32000
#define RETURN 0xD
#define MAXSPEC 25 /*maximum number of spectrum points used by a cell*/
#define MAXCOMPS 4 /*maximum number of components in a cell*/
#define MAXSTR 30 /*maximum component string length*/
#define MAXTRAIN 20 /*maximum number of training files*/
#define NCP1 21 /*MAXTRAIN + 1*/

/* In addition to the general purpose defined constants listed above, */
/* the control of the positioner makes use of various status and control*/
/* bit masks. Pages 22-30 of the PC23-03 manual describe the functions of*/
/* each registers bits (with the exception of the status registers bit 7*/
/* which indicates if the PC23 buffer is over half full). Page 30 gives*/
/* a step by step procedure of the handshaking process. Page 62 explains*/
/* the differences between the binary write (time distance) function and*/
/* the normal write function.                                          */

#define FAIL 0x20 /* 0010 0000 */
#define START_M 0x7F /* 0111 1111 */
#define INTCLR 0x20 /* 0010 0000 */
#define RESTART 0x17 /* 0001 0111 */
#define POSITIVE 0x8000 /* 1000 0000 0000 0000 */
#define ALDONE 0x7 /* 0111 */
#define STOP 0x64 /* 0110 0100 */
#define START 0x40 /* 0100 0000 */
#define CB 0x60 /* 0110 0000 */
#define CB1 0x61 /* 0110 0001 */
#define IDB_M 0x10 /* 0001 0000 */
#define IDB 0x70 /* 0111 0000 */
#define IDB1 0x71 /* 0111 0001 */
#define ODB 0x8 /* 1000 */
#define ACK 0xE0 /* 1110 0000 */
#define BFULL 0x80 /* 1000 0000 */ /*checks buffer status*/

/* All preceding compiler directives are overhead. The actual program*/
/* starts here. No global variables have been declared.              */

main()
{
        /* declarations and initializations */

        int k;/*loop counter*/
        int l;/*axis pointer*/
        int j;/*array pointer -- cell number*/
        int modeflag = TRUE;/*1 => both positioners went home */
        int goflag = TRUE;/*1 => both positioners initialized */
        int quitflag = FALSE;/* TRUE => halt, return to DOS */
        int retflag;/*TRUE => program ran ok*/
        char s[80];/*key input buffer*/
        char string_array[5][MAXCOMPS][MAXSTR];/*composition strings*/
        int cellflag[5];/*TRUE => cell is active*/
        int scans[5];/*number of scans used for current cell*/
        int scount[5];/*number of points in spectrum of current cell*/
```

```
    float norm[5];/*normalization indicator for current cell*/
    int spect[5][MAXSPEC];/*spectrum wave numbers for current cell*/
    int a = TRUE;/*TRUE => all files opened and had good data*/
    int g[5] = {0,0,0,0,0};/*number of components in this cell's mixture*/
    long cellarray0[5][3];/*excite positioner coordinate vectors*/
    long cellarray1[5][3];/*receive positioner coordinate vectors*/
    FILE *ff,*fopen();/*file handling pointers*/
    float spectrum0[MAXSPEC];/*spectrum vector*/
    float comp0[MAXCOMPS];/*composition vector*/
    float peak = 1.0;/*peak value of spectrum*/
    float normfact = 1.0;/*height drift normalizing factor*/
    char er_byte[3];/*error byte*/
    int b;/*number of characters sent to output file this pass*/
    int z;/*TRUE => spectrum collected ok*/
    float f[5][MAXSPEC][MAXTRAIN];/*eigen spectra matrix*/
    int nc[5];/*number of training spectra*/
    float lambda[5][MAXTRAIN];/*eigenvalue vectors*/
    int np[5][MAXCOMPS];/*L2 retention count*/
    int nloadp[5][NCP1][MAXCOMPS];/*load retention matrix*/
    float beta[5][NCP1][MAXCOMPS];/*regression factors*/

    /* the function starts doing work here */

/* pos_there(&goflag); */ /* check whether positioners are there */
    if (goflag)
     {
 /*  set_mode(&modeflag); */ /*send positioners home*/
     goflag = goflag && modeflag;
     }
    if (goflag)
     {
      a = do_files(cellflag,cellarray0,cellarray1,g,string_array,
        scans,scount,norm,spect,f,nc,lambda,np,nloadp,beta);
        /*get data from files*/
      goflag = goflag && a;
     }
    if (goflag)
     {
      ff = fopen("messages.dat","a");/*open output file*/
      if (ff == NULL)
       {
        goflag = FALSE;
        printf("\nOutput file failed to open\n");
       }
     }
    if (goflag)
     {
      a = gig1700(); /*gig the spectrometer*/
      goflag = goflag && a;
     }

    if (goflag) /*if everything is ready*/
     {
      j = 0;/*initialize cell pointer*/
      while(!quitflag)
       {
        if (cellflag[j]) /*bypass inactive cells*/
         {
     /*   flip_dump(1); */ /*move beam dump to block laser*/
     /*   move_stage(j,ADDRESS0,cellarray0); */ /*excitation stage*/
     /*   flip_dump(0); */  /*move beam dump to unblock laser*/
     /*   move_stage(j,ADDRESS1,cellarray1); */ /*receive stage*/
         z = get_spectrum(j,scount[j],spect,scans[j],norm[j],
                          normfact,spectrum0);
         composition(beta,nloadp,np,lambda,f,nc[j],spectrum0,g[j],
```

```
                    scount[j],j,comp0,&peak,norm[j],&normfact);
            erbyte(er_byte);
            b = write_log(ff,j,er_byte,peak,g[j],string_array,comp0);
           }
          printf("\nEnd of cycle\n");
          read_a_line(&quitflag);/*ctrl-X to quit to DOS*/
          j++;
          j = j % 5;
         }
        a = ungig1700();/*return spectrometer to conventional mode*/
        fclose(ff);/*close output file*/
        retflag = TRUE;
       }
      else
       retflag = FALSE;
      return(retflag);

}


/* First Level - Functions First Called by  Main Functions              */

/* function flip_dump()                                                 */
/*                                                                      */
/* Function to operate the beam dump solenoid.                          */
/* The beam dump solenoid is controlled by the logic state on pin 2 of the*/
/* parallel printer port. A byte is written to the port address 378 hex.  */
/* States of pins 2 through 9 on the printer port connector correspond to */
/* the states of bits 0 through 7 resectively of the output byte. Sending */
/* a 0 to the port latches pin 2 low. Sending a 1 to the port latches pin */
/* 2 high.                                                              */
/*                                                                      */
/* CAUTION: DO NOT USE THIS FUNCTION IF THERE IS A PRINTER CONNECTED TO   */
/* THE PARALLEL PRINTER ADAPTER.                                        */
/*                                                                      */
/* input data (parameter):                                              */
/*  flag - 1 => energize pin 2                                          */
/*                                                                      */
/* output data (sent to printer port):                                  */
/*  flag - 1 => energize pin 2                                          */
/*                                                                      */
/* functions called: none                                               */
/************************************************************************/

flip_dump(flag)
int flag;/*1 => energize pin 2*/
{
        outp(ADDRESS2,flag);
}

/* function erbyte()                                                    */
/*                                                                      */
/* function generates error data as ascii representation of hex byte    */
/*                                                                      */
/* In the initial implementation of the cycle program, the hardware is  */
/* checked at the beginning of the run, but not thereafter. Operating   */
/* experience during the second half of phase 2 will indicate what      */
/* conditions, for example, "no flow into sample cell",                 */
/* need to be monitored and trapped. These operating conditions will set */
/* or clear flags. The flag conditions will be reported to the process  */
/* control system as a byte. The byte is reported as an ascii           */
/* representation of a hex byte where each bit is a condition flag.     */
/* The future implementation of this function will collect the flags for */
/* a given pass and compose the error byte from them. The present       */
/* implementation is a rough-in. It generates a fixed dummy byte that can */
```

```
/* be used by other functions.                                              */
/*                                                                          */
/* input data: none yet                                                     */
/*                                                                          */
/* output data (parameter):                                                 */
/*  er_byte - error byte                                                    */
/*                                                                          */
/* functions called: none                                                   */
/*                                                                          */
/**************************************************************************/

erbyte(er_byte)
char er_byte[];/*error byte*/
{
        char c[3] = {"FF"};/*dummy error byte*/

        strcpy(er_byte,c);
        return(1);
}

/* function write_log()                                                     */
/*                                                                          */
/* function makes up composition message and sends it to output file        */
/*                                                                          */
/* input data:                                                              */
/*  obtained from operating system:                                         */
/*   time                                                                   */
/*   date                                                                   */
/*  passed in as parameters:                                                */
/*   cell number                                                            */
/*   error byte                                                             */
/*   spectrum peak                                                          */
/*   number of components                                                   */
/*   component string array                                                 */
/*   composition vector                                                     */
/*                                                                          */
/* output data:                                                             */
/*  composition message - written to output file                            */
/*                                                                          */
/* functions called:                                                        */
/*  ftoa - converts float to ascii string                                   */
/*                                                                          */
/**************************************************************************/

write_log(fr,cell,er_flag,peak,n_comps,sarray,compvec)
FILE *fr;/*pointer to output file*/
int cell;/*sample cell number*/
char er_flag[];/*error byte*/
float peak;/*peak value of spectrum*/
int n_comps;/*number of components in this cell's mixture*/
char sarray[][MAXCOMPS][MAXSTR];/*component string array*/
float compvec[];/*composition vector*/

{
        /*local declarations and initializations*/
        char date_buffer[9];/*date string*/
        char time_buffer[9];/*time string*/
        char scell[17];/*cell number string*/
        char s_comps[17];/*composition count string*/
        char speak[30];/*spectrum peak string*/
        char sbuf[81];/*string transfer buffer*/
        char lstring[MAXCOMPS][MAXSTR];/*local component string buffer*/
        char cstring[MAXCOMPS][81];/*composition string array*/
        char message[512];/*message string*/
        int i;/*loop counter*/
```

```
        int b;/*number of characters transferred to file*/

        /*function starts working here*/
        _strdate(date_buffer);
        _strtime(time_buffer);
        itoa(cell,scell,10);
        ftoa(peak,speak);
        itoa(n_comps,s_comps,10);
        for(i=0;i<n_comps;i++)
         {
          strcpy(sbuf,(&sarray[cell][0][0]) + (MAXSTR*i));
          strcpy((&lstring[0][0]) + (MAXSTR*i),sbuf);
          ftoa(compvec[i],sbuf);
          strcpy((&cstring[0][0]) + (81*i),sbuf);
         }
        strcpy(message,date_buffer);
        strcat(message," ");
        strcat(message,time_buffer);
        strcat(message," ");
        strcat(message,scell);
        strcat(message," ");
        strcat(message,er_flag);
        strcat(message," ");
        strcat(message,speak);
        strcat(message," ");
        strcat(message,s_comps);
        strcat(message," ");
        for (i=0;i<n_comps;i++)
         {
          strcpy(sbuf,(&lstring[0][0]) + (MAXSTR*i));
          strcat(message,sbuf);
          strcat(message," ");
          strcpy(sbuf,(&cstring[0][0]) + (81*i));
          strcat(message,sbuf);
          strcat(message," ");
         }
        strcat(message,"AR ");
        printf("\nOutput string\n");
        printf("\n%s\n",message);
        b = fprintf(fr,"%s\n",message);
        return(b);
}

/* function get_spectrum()                                                  */
/*                                                                          */
/* function collects a spectrum from the interferometer                     */
/*                                                                          */
/* function sends the number of scans to the interferometer, starts the     */
/* scanning operation, polls the interferometer until either the scanning   */
/* is completed or the TIMEOUT constant (seconds) is exceeded, downloads    */
/* the collected spectrum to memory, selects the desired spectrum points,   */
/* drift normalizes the spectrum points if necessary, and returns the       */
/* short spectrum vector to the calling function.                           */
/*                                                                          */
/* NOTE: ACTUAL IMPLEMENTATION OF INTERFEROMETER FUNCTIONS AWAITS FURTHER   */
/* TECHNICAL INFORMATION FROM PERKIN-ELMER. FOR THE MEAN TIME, DUMMY        */
/* VERSIONS OF THE FUNCTIONS HAVE BEEN IMPLEMENTED.                         */
/* THIS ONE READS A SPECTRUM FROM A FILE TO PROVIDE DATA FOR TESTING        */
/* THE COMPOSITION PREDICTION FUNCTION                                      */
/*                                                                          */
/* input data:                                                              */
/*  parameters:                                                             */
/*   cell - cell number                                                     */
/*   cols - number of spectrum points to be returned                        */
/*   spect - list of wave numbers for which intensities are sought          */
```

```
/*   scans - number of scans to be used in collection of spectrum        */
/*   normflag - >0.1 => normalize for height drift                       */
/*   normfact - multiply each point by this factor if drift normalizing  */
/*  from interferometer via GPIB bus:                                    */
/*   long_spec - long version of spectrum                                */
/*                                                                       */
/* output data:                                                          */
/*  parameter: spectrum0 - short version of normalized spectrum vector   */
/*  call return: c - TRUE => spectrum collected OK                       */
/*                                                                       */
/* functions called: none yet                                            */
/*                                                                       */
/*************************************************************************/

get_spectrum(cell,cols,spect,scans,normflag,normfact,spectrum0)
int cell;/*cell number*/
int cols;/*number of components in spectrum*/
int spect[][MAXSPEC];/*wave numbers for which intensities are sought*/
int scans;/*number of scans in spectrum*/
float normflag;/* >0.1 => normalize for height drift*/
float normfact;/*normalizing factor*/
float spectrum0[];/*spectrum vector*/

{
        int k;/*loop counter*/
        int c = TRUE;/*TRUE => file read ok*/
        int d;/*count of correct readings*/
        float cx;/*input buffer*/
        FILE *fk,*fopen();/*file handling pointers*/

        fk = fopen("newspec.dat","r");
        if (fk != NULL)  /*if file opened */
            {
                for (k=0;((k<cols) && c);k++)
                 {
                  d = fscanf(fk,"%f",&cx); /*nead next number to buffer*/
                  if (d == 1) /*if it worked write to matrix */
                   spectrum0[k] = cx;
                  else
                   {
                    printf("\nError reading spectrum file\n");
                    c = FALSE;
                   }
                 }

              fclose(fk);
            }
        else
            {
             printf("\nSpectrum file did not open\n");
             c = FALSE;
            }
        return(c);
}

/*function composition()                                                 */
/*                                                                       */
/* returns peak of spectrum and composition vector                       */
/*                                                                       */
/* input data (passed in as parameters):                                 */
/*  beta - regression matrix                                             */
/*  nloadp - retention matrix                                            */
/*  np - retention count vector                                          */
/*  lambda - eigenvalues of z                                            */
/*  f - eigenspectra of z                                                */
```

```
/*  nc - number of training samples                                        */
/*  spectr - spectrum vector                                               */
/*  nprop - number of components                                           */
/*  nr - number of spectrum points                                         */
/*  cell - cell number                                                     */
/*  expected - expected calibration value                                  */
/*                                                                         */
/* output data (passed out as parameters):                                 */
/*  compo - composition vector                                             */
/*  norm - normalizing factor                                              */
/*  peak - peak value of spectrum                                          */
/*                                                                         */
/* functions called:                                                       */
/*  frmul2 - form loading matrix                                           */
/*  frmul3 - form correlated loading matrix from loading matrix            */
/*  estup3 - estimate composition from correlated loading matrix           */
/*                                                                         */
/*************************************************************************/

composition(beta,nloadp,np,lambda,f,nc,spectr,nprop,nr,cell,compo,peak,
            expected,normf)
float beta[][NCP1][MAXCOMPS];/*regression matrix*/
int nloadp[][NCP1][MAXCOMPS];/*retention matrix*/
int np[][MAXCOMPS];/*retention count vector*/
float lambda[][MAXTRAIN];/*eigenvalues of z*/
float f[][MAXSPEC][MAXTRAIN];/*eigenspectra of z*/
int nc;/*number of training samples*/
float spectr[];/*spectrum vector*/
int nprop;/*number of components*/
int nr;/*number of spectrum points*/
int cell;/*cell number*/
float expected;/*expected calibration value*/
float compo[];/*composition vector*/
float *normf;/*normalizing factor*/
float *peak;/*peak value of spectrum*/
{
        /*local declarations and initializations*/
        int i;/*loop counter*/
        float ul2[NCP1];/*loadings*/
        float ul3[NCP1][MAXCOMPS];/*correlated loadings*/

        /*function starts working here*/
        *peak = spectr[0];
        for (i=1;i<nr;i++)
         if (spectr[i] > *peak)
          *peak = spectr[i];
        if (nprop == 1)
         {
          compo[0] = 100.0;
          *normf = expected/(*peak);
         }
        else
         {
          frmul2(nr,nc,spectr,cell,f,lambda,ul2);
          frmul3(cell,nprop,np,ul2,nloadp,ul3);
          estup3(cell,nprop,np,ul3,beta,compo);
         }
}


/* function pos_there                                                      */
/*                                                                         */
/* initializes both positioners and makes sure they are both there         */
/*                                                                         */
/* input data:                                                             */
```

```
/*  readflag - flag passed back by initialize()                          */
/*                                                                       */
/* output data:                                                          */
/*  gofrag - flag passsed back to calling function                       */
/*                                                                       */
/* functions called:                                                     */
/*  initialize() - initializes addressed positioner                      */
/*                                                                       */
/***********************************************************************/

pos_there(gofrag)
int *gofrag;/*TRUE => Both positioners initialized*/

{
        int readflag = TRUE;/* TRUE => positioner initialized */
        int address;/*address of positioner*/

        address = ADDRESS0;
        initialize(address,&readflag);/*set up PC23*/
        *gofrag = readflag;
        if (!readflag)
         printf("\nPositioner %x failed to initialize\n",address);
        address = ADDRESS1;
        initialize(address,&readflag);
        *gofrag = *gofrag && readflag;
        if (!readflag)
         printf("\nPositioner %x failed to initialize\n",address);


}


/* function set_mode                                                     */
/*                                                                       */
/* homes both positioners and sets to absolute positioning mode          */
/*                                                                       */
/* input data:                                                           */
/*  readflag - flag passed back by set_one()                             */
/*                                                                       */
/* output data:                                                          */
/*  gofrag - flag passsed back to calling function                       */
/*                                                                       */
/* functions called:                                                     */
/*  set_one() - send messages to addressed positioner                    */
/*                                                                       */
/***********************************************************************/

set_mode(gofrag)
int *gofrag;/*TRUE => Both positioners went home*/

{
        int readflag = TRUE;/* TRUE => positioner went home */
        int address;/*address of positioner*/

        address = ADDRESS0;
        set_one(address,&readflag);/*set up PC23*/
        *gofrag = readflag;
        if (!readflag)
         printf("\nPositioner %x failed to go home\n",address);
        address = ADDRESS1;
        set_one(address,&readflag);
        *gofrag = *gofrag && readflag;
        if (!readflag)
         printf("\nPositioner %x failed to go home\n",address);
```

```
}

/* function gig1700()                                                    */
/*                                                                       */
/* function to set up the interferometer                                 */
/*                                                                       */
/* function causes the control computer to assume control of the IEEE-488 */
/* bus, set up the interferometer, set it to ASCII mode, and query for an */
/* acknowledgment from the interferometer.                               */
/*                                                                       */
/* NOTE: ACTUAL IMPLEMENTATION OF INTERFEROMETER FUNCTIONS AWAITS FURTHER */
/* TECHNICAL INFORMATION FROM PERKIN-ELMER. FOR THE MEAN TIME, DUMMY     */
/* VERSIONS OF THE FUNCTIONS HAVE BEEN IMPLEMENTED.                      */
/*                                                                       */
/* Input data: none                                                      */
/*                                                                       */
/* Output data:                                                          */
/*  z - TRUE => interferometer responded as expected                     */
/*                                                                       */
/* Functions called: none                                                */
/*                                                                       */
/*************************************************************************/

gig1700()
{
        return(TRUE);
}

/* function ungig1700()                                                  */
/*                                                                       */
/* function to return interferometer to conventional mode                */
/*                                                                       */
/* function causes the control computer to relinquish control of the GPIB */
/* bus, set interferometer to binary mode, and return control to the     */
/* interferometer controller keyboard.                                   */
/*                                                                       */
/* NOTE: ACTUAL IMPLEMENTATION OF INTERFEROMETER FUNCTIONS AWAITS FURTHER */
/* TECHNICAL INFORMATION FROM PERKIN-ELMER. FOR THE MEAN TIME, DUMMY     */
/* VERSIONS OF THE FUNCTIONS HAVE BEEN IMPLEMENTED.                      */
/*                                                                       */
/* Input data: none                                                      */
/*                                                                       */
/* Output data:                                                          */
/*  z - TRUE => interferometer responded as expected                     */
/*                                                                       */
/* Functions called: none                                                */
/*                                                                       */
/*************************************************************************/

ungig1700()
{
        return(TRUE);
}

/* function move_stage()                                                 */
/*                                                                       */
/* Executes absolute translation of stage in 3 dimensions                */
/*                                                                       */
/* Before this function is invoked, other functions have initialized the */
/* positioner and send commands setting the home position, velocity,     */
/* and acceleration. The mode has been preset to absolute displacement   */
/* with respect to home coordinates.                                     */
/*                                                                       */
/* The function reveives a copy of the cell number, the motor controller */
/* address, and a pointer to the to the array of coordinates, from the   */
```

```
/* calling function. It selects the three appropriate coordinates from the */
/* array, formulates the displacement strings, sends the strings to the    */
/* positioner, and sends the "go" message to the positioner.               */
/*                                                                         */
/* input data (parameters from calling function):                          */
/*  jj - cell number                                                       */
/*  addr - I/O address of motor controller                                 */
/*  coords - array of positioniing coordinates                             */
/*                                                                         */
/* output data (parameter to called function):                             */
/*  s - string sent to positioner via writecmd()                           */
/*  Note: actual output is new location of positioner                      */
/*                                                                         */
/* functions called:                                                       */
/*  writecmd() - sends string to positioner                                */
/*                                                                         */
/*************************************************************************/

move_stage(jj,addr,coords)
int jj;/*cell number*/
int addr;/*I/O address of motor controller*/
long coords[][3];/*list of coordinates*/

{
        char s[81];/*string sent to controller*/
        int i;/*loop counter*/
        char s1[17];/*axis portion of string*/
        char s2[2] = "D";/*command portion of string*/
        char s3[17];/*actual displacement in motor steps*/
        char s4[81];/*positioner response buffer*/
        long temp;/*temporary coordinate placeholder*/

        for (i=1;i<=3;i++) /*i is axis number */
         {
          temp = coords[jj][i-1];
          ltoa(temp,s3,10);
          itoa(i,s1,10);
          strcpy(s,s1);
          strcat(s,s2);
          strcat(s,s3);
          strcat(s," ");
          printf("\nString %s sent to positioner\n",s);
          writecmd(addr,s);  /*send displacement to positioner*/
          }
        strcpy(s,"G123 ");
        printf("\nString %s sent to positioner\n",s);
        writecmd(addr,s); /*make the stage go*/
        writecmd(addr," 1PR ");
        readanswer(addr,s4);
        printf("\nAxis 1 is at %s\n",s4);
        writecmd(addr," 2PR ");
        readanswer(addr,s4);
        printf("\nAxis 2 is at %s\n",s4);
        writecmd(addr," 3PR ");
        readanswer(addr,s4);
        printf("\nAxis 3 is at %s\n",s4);
}


/* function read_a_line()                                                  */
/*                                                                         */
/* polls key buffer to see if Ctrl-x has been hit                          */
/* function return a TRUE if Ctrl-x is hit                                 */
/*                                                                         */
/* input data:                                                             */
```

```
/*  go_flag - string furnished by user                                    */
/*                                                                        */
/* output data (returned as parameters):                                  */
/*  quitfrag - TRUE => return to DOS                                      */
/*                                                                        */
/* functions called: none                                                 */
/**************************************************************************/

read_a_line(quitfrag)
int *quitfrag;/*TRUE => return to DOS*/

{
        /*local declarations*/
        char go_flag;/*first letter of message*/

        /*function starts here*/
        printf("\n");
        printf("Hit Ctrl-X to Quit: ");
        *quitfrag = FALSE;
        if ( kbhit() )
         {
          go_flag = getche();
          if (go_flag == ('X'-0X40))
                *quitfrag = TRUE;
          else
                *quitfrag = FALSE;
         }
        fflush(stdin);
}



/* function do_files()                                                    */
/*                                                                        */
/* function initializes data arrays from external files                   */
/*                                                                        */
/* Input data (parameter arrays received from lower level functions):     */
/*  f1 = cell status vector -- TRUE => cell is active                     */
/*  a0 = excitation positioner coordinate vectors                         */
/*  a1 = receiving positioner coordinate vectors                          */
/*  g = number of components in this cell's mixture                       */
/*  string_array = composition strings                                    */
/*  scans = number of scans used for current cell                         */
/*  scount = number of points in spectrum of current cell                 */
/*  norm = normalization indicator for current cell                       */
/*  spect = spectrum wave numbers for current cell                        */
/*  f = eigen spectra matrix                                              */
/*  nc = number of training spectra                                       */
/*  lambda = eigenvalue vectors                                           */
/*  np = L2 retention count                                               */
/*  nloadp = load retention matrix                                        */
/*  beta = regression factors                                             */
/*                                                                        */
/* Output data (parameter arrays sent to calling function):               */
/*  f1 = cell status vector -- TRUE => cell is active                     */
/*  a0 = excitation positioner coordinate vectors                         */
/*  a1 = receiving positioner coordinate vectors                          */
/*  g = number of components in this cell's mixture                       */
/*  string_array = composition strings                                    */
/*  scans = number of scans used for current cell                         */
/*  scount = number of points in spectrum of current cell                 */
/*  norm = normalization indicator for current cell                       */
/*  spect = spectrum wave numbers for current cell                        */
/*  f = eigen spectra matrix                                              */
/*  nc = number of training spectra                                       */
```

```
/*  lambda = eigenvalue vectors                                          */
/*  np = L2 retention count                                              */
/*  nloadp = load retention matrix                                       */
/*  beta = regression factors                                            */
/*                                                                       */
/* Functions called:                                                     */
/*  cellfunc() - brings in data from cell files                          */
/*  feedback1() - prints cell file data to screen                        */
/*  matfunc() - brings in data from matrix files                         */
/*  feedback2() - prints matrix file data to screen                      */
/*                                                                       */
/***********************************************************************/

do_files(f1,a0,a1,g,string_array,scans,scount,norm,spect,f,nc,lambda,np,
         nloadp,beta)
int f1[];/*TRUE => cell is active*/
long a0[][3];/*excite positioner coordinate vectors*/
long a1[][3];/*receive positioner coordinate vectors*/
int g[];/*number of components in this cell's mixture*/
char string_array[][MAXCOMPS][MAXSTR];/*composition strings*/
int scans[];/*number of scans used for current cell*/
int scount[];/*number of points in spectrum of current cell*/
float norm[];/*normalization indicator for current cell*/
int spect[][MAXSPEC];/*spectrum wave numbers for current cell*/
float f[][MAXSPEC][MAXTRAIN];/*eigen spectra matrix*/
int nc[];/*number of training spectra*/
float lambda[][MAXTRAIN];/*eigenvalue vectors*/
int np[][MAXCOMPS];/*L2 retention count*/
int nloadp[][NCP1][MAXCOMPS];/*load retention matrix*/
float beta[][NCP1][MAXCOMPS];/*regression factors*/

{
        int a;/*TRUE => cell files ok*/
        int b;/*TRUE => matrix files ok*/
        int c;/*TRUE => data moved ok from files to arrays*/

        a = cellfunc(f1,a0,a1,g,string_array,scans,scount,norm,spect);
        feedback1(f1,a0,a1,g,string_array,a,scans,scount,norm,spect);
        if (a)
         {
          b = matfunc(f,f1,g,scount,nc,lambda,np,nloadp,beta);
          feedback2(f,nc,lambda,np,nloadp,beta);
         }
        c = a && b;
        return(c);
}

/* Second Level - Functions First Called by First Level Functions       */

/* function frmul2                                                      */
/*                                                                      */
/* Function forms the loading from the spectrum of the unknown          */
/* sample. This is the first of three steps in deducing the             */
/* composition of the sample from its spectrum.                         */
/*                                                                      */
/* Loading is calculated by matrix multiplication of the transpose      */
/* of the eigenspectra matrix by the spectrum vector. This result       */
/* then post multiplies the inverted eigenvalue matrix. Since the       */
/* eigenvalue matrix is diagonal, its inverse is a diagonal matrix      */
/* of the reciprocals of the original diagonal elements.                */
/*                                                                      */
/* Input data (parameters):                                             */
/*  nr - points in spectrum; also rows in f                             */
/*  nc - number of training samples; also columns in f                  */
/*  h - cell number                                                     */
```

```
/*  spectrum - spectrum to be analyzed, has nr elements              */
/*  f - eigenspectra array; eigenvectors of product moment matrix    */
/*  lambda - eigenvalues of product momemt matrix                    */
/*                                                                   */
/* Output data (parameter):                                          */
/*  ul2 - loading of spectrum of unknown sample, has nc+1 elements   */
/*                                                                   */
/* Functions called: none                                            */
/*                                                                   */
/*********************************************************************/

frmul2(nr,nc,spectrum,h,f,lambda,ul2)
int nr;/*points in spectrum; also rows in f*/
int nc;/*number of training samples; also columns in f*/
int h;/*cell number*/
float spectrum[];/*spectrum to be analyzed, has nr elements*/
float f[][MAXSPEC][MAXTRAIN];/*eigenspectra array*/
float lambda[][MAXTRAIN];/*eigenvalues of product momemt matrix*/
float ul2[];/*loading of unknown spectrum, has nc+1 elements*/

{
        /*local declarations and initializations*/
        int j;/*points to a position in the ul2 vector*/
        int k;/*points to elements to be multiplied*/
        int cccp;/*limiting length of load vector*/

        /*function starts doing work here*/
        cccp = nc+1;
        ul2[0] = 1.0;/*augment the loading*/
        for (j=1;j<cccp;j++)/*step thru elements of ul2*/
         {
          ul2[j] = 0.0;/*initialize jth element*/
          for (k=0;k<nr;k++)/*dot product of jth column of f and spectrum*/
           ul2[j] = ul2[j] + (f[h][k][j-1] * spectrum[k]);
          ul2[j] = ul2[j]/lambda[h][j-1];
         }
}

/* function frmul3                                                   */
/*                                                                   */
/* Function selects loadings from ul2 which correlate with properties*/
/* of the training samples and moves them to ul3.                    */
/* This is the second of three steps in deducing the                 */
/* composition of the sample from its spectrum.                      */
/*                                                                   */
/* For each component the correlations are different. Thus for each  */
/* component, different loadings must be retained. Hence, whereas ul2*/
/* is a vector or row matrix, ul3 is a two dimensional matrix, where */
/* each row corresponds to a component                               */
/*                                                                   */
/* Input data (parameters):                                          */
/*  h - cell number                                                  */
/*  nprop - number of different components in the mixture            */
/*  np - retention count vector                                      */
/*  nloadp - matrix of specific retentions                           */
/*  ul2 - loading of spectrum of unknown sample, has nc+1 elements   */
/*                                                                   */
/* Output data (parameter):                                          */
/*  ul3p - correlated loading matrix                                 */
/*                                                                   */
/* Functions called: none                                            */
/*                                                                   */
/*********************************************************************/

frmul3(h,nprop,np,ul2,nloadp,ul3p)
```

```
int h;/*cell number*/
float ul2[];/*loading of unknown spectrum, has nc+1 elements*/
int nprop;/*number of different components in the mixture*/
int np[][MAXCOMPS];/*retention count vector*/
int nloadp[][NCP1][MAXCOMPS];/*matrix of specific retentions*/
float ul3p[][MAXCOMPS];/*correlated loading matrix*/

{
        /*local declarations and initializations*/
        int iprop;/*points to current property*/
        int j;/*points at position in np vector*/
        int jj;/*points at desired position in ul2 vector*/

        /*function starts doing work here*/
        for (iprop=0;iprop<nprop;iprop++)
         for (j=0;j<np[h][iprop];j++)
          {
           jj = nloadp[h][j][iprop] - 1;
           ul3p[j][iprop] = ul2[jj];
          }
}

/* function estup3                                                    */
/*                                                                    */
/* Function calculates concentrations of components of unknown        */
/* mixture by multiplying loading matrix by regression matrix.        */
/* This is the third of three steps in deducing the                   */
/* composition of the sample from its spectrum.                       */
/*                                                                    */
/* Input data (parameters):                                           */
/*  h - cell number                                                   */
/*  nprop - number of different components in the mixture             */
/*  np - retention count vector                                       */
/*  ul3p - correlated loading matrix                                  */
/*  b3prop - regression matrix                                        */
/*                                                                    */
/* Output data (parameter):                                           */
/*  upest - composition vector for spectrum of unknown sample         */
/*                                                                    */
/* Functions called: none                                             */
/*                                                                    */
/**********************************************************************/

estup3(h,nprop,np,ul3p,b3prop,upest)
int h;/*cell number*/
int nprop;/*number of different components in the mixture*/
int np[][MAXCOMPS];/*retention count vector*/
float ul3p[][MAXCOMPS];/*correlated loading matrix*/
float b3prop[][NCP1][MAXCOMPS];/*regression matrix*/
float upest[];/*composition vector*/

{
        /*local declarations and initializations*/
        int iprop;/*points to current property*/
        int j;/*points at positions to be multiplied*/

        /*function starts doing work here*/
        for (iprop=0;iprop<nprop;iprop++)
         {
          upest[iprop] = 0.0;
          for (j=0;j<np[h][iprop];j++)
           upest[iprop] = upest[iprop] + ul3p[j][iprop]*b3prop[h][j][iprop];
         }
}
```

```
/*function ftoa()                                                          */
/*                                                                        */
/* converts floating point number to E format string                      */
/*                                                                        */
/* input data (parameter):                                                */
/*  x - floating point number                                             */
/*                                                                        */
/* output data (paramater):                                               */
/*  s - string version of floating point number                           */
/*                                                                        */
/* functions called: (none)                                               */
/*                                                                        */
/**************************************************************************/

ftoa(x,s)
float x;/*floating point number to be converted*/
char s[];/*string to be returned*/
{
        double y;/*type change place holder*/
        int dec;/*decimal point in conversion*/
        int sign;/*sign of floating point value*/
        int precision=10;/*number of digits to be converted*/
        char *speak;/*pointer to converted string*/
        char s1[15];/*temporary string holder*/
        char sdec[17];/*string version of sign*/
        int i;/*loop counter*/
        int j;/*string pointer*/


        y = (double) x;
        speak = ecvt(y,precision,&dec,&sign);

        itoa(dec,sdec,10);
        if (sign == 0)
         strcpy(s,"+0.");
        else
         strcpy(s,"-0.");
        j = 0;
        while (*speak != '\0')
         s1[j++] = *speak++;
        s1[j] = '\0';
        strcat(s,s1);
        strcat(s,"E");
        strcat(s,sdec);

        return(1);
}


/* function matfunc()                                                     */
/*                                                                        */
/* function initializes data arrays from matrix files                     */
/*                                                                        */
/* Input data (data in ascii files matrix0.dat through matrix5.dat):      */
/*  f = eigen spectra matrix                                              */
/*  nc = number of training spectra                                       */
/*  lambda = eigenvalue vectors                                           */
/*  np = L2 retention count                                               */
/*  nloadp = load retention matrix                                        */
/*  beta = regression factors                                             */
/*                                                                        */
/* Output data (parameter array sent to calling function):                */
/*  f = eigen spectra matrix                                              */
/*  nc = number of training spectra                                       */
/*  lambda = eigenvalue vectors                                           */
```

```
/*  np = L2 retention count                                                 */
/*  nloadp = load retention matrix                                          */
/*  beta = regression factors                                               */
/*                                                                          */
/* Functions called: none                                                   */
/*                                                                          */
/**************************************************************************/

matfunc(f,active,comps,spects,nc,lambda,np,nloadp,beta)
int active[];/*TRUE => cell active*/
int comps[];/*number of components in cell*/
int spects[];/*number of points in cell spectrum*/
float f[][MAXSPEC][MAXTRAIN];/*eigen spectra matrix*/
int nc[];/*number of training spectra*/
float lambda[][MAXTRAIN];/*eigenvalue vectors*/
int np[][MAXCOMPS];/*L2 retention count*/
int nloadp[][NCP1][MAXCOMPS];/*load retention matrix*/
float beta[][NCP1][MAXCOMPS];/*regression factors*/

{
        /*local declarations and initializations*/

        static char mat_list[5][13] = {
                "matrix0.dat",
                "matrix1.dat",
                "matrix2.dat",
                "matrix3.dat",
                "matrix4.dat"
                    };/*files containing matrix data*/
        char matt[13];/*current matrix file*/
        int i,j,k;/*loop counters*/
        int h;/*cell number*/
        int c = TRUE;/*TRUE => matrix data obtained from files*/
        int d;/*conversion counter*/
        float cx;/*input buffer*/
        int anc;/*input buffer*/
        FILE *fr,*fopen();/*file handling pointers*/

        /*function starts working here*/
        /*initialize matrix space to all zeros*/

        for (i=0;i<5;i++)
         nc[i] = 0;
        for (i=0;i<5;i++)
         for (j=0;j<MAXSPEC;j++)
          for (k=0;k<MAXTRAIN;k++)
           f[i][j][k] = 0.0;
        for (i=0;i<5;i++)
         for (j=0;j<MAXTRAIN;j++)
          lambda[i][j] = 0.0;
        for (i=0;i<5;i++)
         for (j=0;j<MAXCOMPS;j++)
          np[i][j] = 0;
        for (i=0;i<5;i++)
         for (j=0;j<NCP1;j++)
          for (k=0;k<MAXCOMPS;k++)
           nloadp[i][j][k] = 0;
        for (i=0;i<5;i++)
         for (j=0;j<NCP1;j++)
          for (k=0;k<MAXCOMPS;k++)
           beta[i][j][k] = 0.0;

        for (h=0;((h<5) && c);h++) /*h is cell number*/
         if (active[h] && (comps[h] != 1)) /*bypass inactive cells*/
          {
```

```
strcpy(matt,(&mat_list[0][0]) + (13*h));
fr = fopen(matt,"r");
if (fr != NULL)  /*if file opened */
 {
   d = fscanf(fr,"%d",&anc);
   if (d == 1)
    nc[h] = anc;
   else
    c = FALSE;
   if (c)
    for (j=0;((j<spects[h]) && c);j++)
     for (k=0;((k<nc[h]) && c);k++)
      {
       d = fscanf(fr,"%f",&cx); /*nead next number to buffer*/
       if (d == 1) /*if it worked write to matrix */
        f[h][j][k] = cx;
       else
        {
         printf("\Error reading cell %d matrix file\n",h);
         c = FALSE;
        }
      }
   if (c)
     for (k=0;((k<nc[h]) && c);k++)
      {
       d = fscanf(fr,"%f",&cx); /*nead next number to buffer*/
       if (d == 1) /*if it worked write to matrix */
        lambda[h][k] = cx;
       else
        {
         printf("\Error reading cell %d matrix file\n",h);
         c = FALSE;
        }
      }
   if (c)
    for (k=0;((k<comps[h]) && c);k++)
      {
       d = fscanf(fr,"%d",&anc); /*nead next number to buffer*/
       if (d == 1) /*if it worked write to matrix */
        np[h][k] = anc;
       else
        {
         printf("\Error reading cell %d matrix file\n",h);
         c = FALSE;
        }
      }
   if (c)
    for (j=0;((j<(nc[h]+1)) && c);j++)
     for (k=0;((k<comps[h]) && c);k++)
      {
       d = fscanf(fr,"%d",&anc); /*nead next number to buffer*/
       if (d == 1) /*if it worked write to matrix */
        nloadp[h][j][k] = anc;
       else
        {
         printf("\Error reading cell %d matrix file\n",h);
         c = FALSE;
        }
      }
   if (c)
    for (j=0;((j<(nc[h]+1)) && c);j++)
     for (k=0;((k<comps[h]) && c);k++)
      {
       d = fscanf(fr,"%f",&cx); /*nead next number to buffer*/
       if (d == 1) /*if it worked write to matrix */
```

```
                beta[h][j][k] = cx;
               else
                {
                 printf("\Error reading cell %d matrix file\n",h);
                 c = FALSE;
                }
              }
             fclose(fr);
            }
           else
            {
             printf("\nMatrix file for cell %d did not open\n",h);
             c = FALSE;
            }
          }
        return(c);
}

/* function cellfunc()                                                    */
/*                                                                        */
/* function initializes data arrays from cell files                       */
/*                                                                        */
/* Input data (data in ascii files cella0.dat through cella5.dat):        */
/*  f = cell status vector -- TRUE => cell is active                      */
/*  a0 = excitation positioner coordinate vectors                         */
/*  a1 = receiving positioner coordinate vectors                          */
/*  g = number of components in this cell's mixture                       */
/*  string_array = composition strings                                    */
/*  scans = number of scans used for current cell                         */
/*  scount = number of points in spectrum of current cell                 */
/*  norm = normalization indicator for current cell                       */
/*  spect = spectrum wave numbers for current cell                        */
/*                                                                        */
/* Output data (parameter arrays sent to calling function):               */
/*  f = cell status vector -- TRUE => cell is active                      */
/*  a0 = excitation positioner coordinate vectors                         */
/*  a1 = receiving positioner coordinate vectors                          */
/*  g = number of components in this cell's mixture                       */
/*  string_array = composition strings                                    */
/*  scans = number of scans used for current cell                         */
/*  scount = number of points in spectrum of current cell                 */
/*  norm = normalization indicator for current cell                       */
/*  spect = spectrum wave numbers for current cell                        */
/*                                                                        */
/* Functions called: none                                                 */
/*                                                                        */
/************************************************************************/

cellfunc(f,a0,a1,g,string_array,scans,scount,norm,spect)
int f[];/*TRUE => cell is active*/
long a0[][3];/*excite positioner coordinate vectors*/
long a1[][3];/*receive positioner coordinate vectors*/
int g[];/*number of components in this cell's mixture*/
char string_array[][MAXCOMPS][MAXSTR];/*composition strings*/
int scans[];/*number of scans used for current cell*/
int scount[];/*number of points in spectrum of current cell*/
float norm[];/*normalization indicator for current cell*/
int spect[][MAXSPEC];/*spectrum wave numbers for current cell*/

{
        /*local declarations and initializations*/
        char *c;/*string input checker*/
        char s[MAXSTR];/*string input buffer*/
        char check[MAXSTR];/*checkout string buffer*/
        int cellok = TRUE;/*TRUE => cell files opened and had good data*/
```

```
        int i;/*loop counter*/
        int d;/*conversion counter*/
        int max;/*place holder*/
        int j;/*current cell number*/
        static char file_list[5][13] = {
                "cella0.dat",
                "cella1.dat",
                "cella2.dat",
                "cella3.dat",
                "cella4.dat"
                        }; /*files containing cell data*/
        char cell[13];/*name of current cell file*/
        FILE *fp,*fopen();/*file handling pointers*/

    /*the function starts doing work here*/
    for (j=0;((j<5) && cellok);j++)
    /*open the jth file in turn*/
    /*copy its data to the jth position in each array*
    /*close each file before opening the next*/
    /*loop does not procede beyond first bad cell*/
     {
        strcpy(cell,(&file_list[0][0]) + (13*j));
        fp = fopen(cell,"r");
        if (fp != NULL) /*if file opened*/
         {
          d = fscanf(fp,"%d",&f[j]);
          if (d == 1) /*if first character was successfully read*/
           {
            if (f[j] == 1) /*if jth cell is active*/
             {
              d = fscanf(fp,"%ld%ld%ld%ld%ld%ld%d%f%d",&a0[j][0],&a0[j][1],
&a0[j][2],&a1[j][0],&a1[j][1],&a1[j][2],&scans[j],&norm[j],&scount[j]);
             if (d == 9) /*if next 9 data points were successfully read*/
              {
               for (i=0;i<3;i++)
                printf("\na0[%d][%d]=%ld a1[%d][%d]=%ld\n",
                 j,i,a0[j][i],j,i,a1[j][i]);
               printf("\nNumber of scans is %d\n",scans[j]);
               printf("\nNormalizing marker is %f\n",norm[j]);
               printf("\nNumber of spectrum points is %d\n",scount[j]);
               if (scount[j] > MAXSPEC)
                {
                 max = MAXSPEC;
                 printf("\nSpectrum points for cell %d > %d\n",j,max);
                 cellok = FALSE;
                }
               else
                for (i=0;((i<scount[j]) && cellok);i++)
                 {
                  d = fscanf(fp,"%d",&spect[j][i]);
                  if (d != 1)
                   {
                    cellok = FALSE;
                    printf("\nSpectrum not right for cell %d\n",j);
                   }
                 }
               if (cellok)
                for (i=0;i<scount[j];i++)
                 printf("\nCell %d spectrum point %d is %d\n",j,i,spect[j][i]);
               if (cellok)
                {
                 d = fscanf(fp,"%d",&g[j]);
                 if (d != 1)
                  {
                   printf("\nComponent count not correctly read\n");
```

```
            cellok = FALSE;
           }
         }
        else
         printf("\nNo attempt to read component count\n");
        if (cellok)
         {
          printf("\nNumber of strings is %d\n",g[j]);
          if (g[j] > MAXCOMPS) /*if there are more than MAXCOMPS components*/
           {
            cellok = FALSE;
            max = MAXCOMPS;
            printf("\nNumber of components in cell %d > %d\n",j,max);
           }
         }
        if (cellok)
         {
          fgets(s,MAXSTR-1,fp); /*read rest of line and ignore*/
          for (i=0;((i<g[j]) && cellok);i++)
           {
            c = fgets(s,MAXSTR-1,fp); /*read next composition line*/
            if ((c == NULL) || (strlen(s) == 0))
             {
              cellok = FALSE;
              printf("\nString not right in cell %d\n",j);
             }
            else
             {
              printf("\nString number %d is %s\n",i,s);
              strcpy((&string_array[j][0][0]) + (MAXSTR*i),s);
             }
           }
         }
        if (cellok)
         for (i=0;i<g[j];i++)
          {
           strcpy(check,(&string_array[j][0][0]) + (MAXSTR*i));
           printf("\nstring i=%d j=%d is %s \n",i,j,check);
          }
       }
      else
       {
        printf("\nfscanf cell %d did not make 9 conversions\n",j);
        cellok = FALSE;
       }
     }
    else
     printf("\ncell %d not active\n",j);
   }
  else
   {
    printf("\nfscanf did not make first conversion in cell %d \n",j);
    cellok = FALSE;
   }
  fclose(fp); /*done with jth file -- close it*/
 }
else
 {
  printf("\nfile %s failed to open\n",cell);
  cellok = FALSE;
 }
}
if (cellok)
 printf("\nInput of data from cell files to arrays went well\n");
else
```

```
     printf("\nInput of data from cell files to arrays went badly\n");
    return(cellok);
}


/* function feedback1()                                                    */
/*                                                                         */
/* function displays data arrays from cell files                           */
/*                                                                         */
/* Input data (passed in as parameters):                                   */
/*  f = cell status vector -- TRUE => cell is active                       */
/*  a0 = excitation positioner coordinate vectors                          */
/*  a1 = receiving positioner coordinate vectors                           */
/*  g = number of components in this cell's mixture                        */
/*  string_array = composition strings                                     */
/*  scans = number of scans used for current cell                          */
/*  scount = number of points in spectrum of current cell                  */
/*  norm = normalization indicator for current cell                        */
/*  spect = spectrum wave numbers for current cell                         */
/*                                                                         */
/* Output data (displayed to screen):                                      */
/*  f = cell status vector -- TRUE => cell is active                       */
/*  a0 = excitation positioner coordinate vectors                          */
/*  a1 = receiving positioner coordinate vectors                           */
/*  g = number of components in this cell's mixture                        */
/*  string_array = composition strings                                     */
/*  scans = number of scans used for current cell                          */
/*  scount = number of points in spectrum of current cell                  */
/*  norm = normalization indicator for current cell                        */
/*  spect = spectrum wave numbers for current cell                         */
/*                                                                         */
/* Functions called: none                                                  */
/*                                                                         */
/*************************************************************************/


feedback1(f,a0,a1,g,string_array,a,scans,scount,norm,spect)
int f[];/*TRUE => cell is active*/
long a0[][3];/*excite positioner coordinate vectors*/
long a1[][3];/*receive positioner coordinate vectors*/
int g[];/*number of components in this cell's mixture*/
char string_array[][MAXCOMPS][MAXSTR];/*composition strings*/
int a;/*TRUE => input data ok*/
int scans[];/*number of scans used for current cell*/
int scount[];/*number of points in spectrum of current cell*/
float norm[];/*normalization indicator for current cell*/
int spect[][MAXSPEC];/*spectrum wave numbers for current cell*/

{
        int i,j;/*loop counters*/
        char check[MAXSTR];/*check string*/

        if (a)
          {
           for (j=0;j<5;j++)
            {
             printf("\nCell %d : In use flag = %d \n",j,f[j]);
             if (f[j] == 1)
              {
                 for (i=0;i<3;i++)
                  printf("\na0[%d][%d]=%ld a1[%d][%d]=%ld\n",
                   j,i,a0[j][i],j,i,a1[j][i]);
                 printf("\nNumber of scans = %d\n",scans[j]);
                 printf("\nNormalization indicator is %f\n",norm[j]);
                 printf("\n %d points used in spectrum\n",scount[j]);
```

```
                for (i=0;i<scount[j];i++)
                 printf("\nSpectrum point %d is %d\n",i,spect[j][i]);
                printf("\nNumber of strings is %d\n",g[j]);
                for (i=0;i<g[j];i++)
                  {
                   strcpy(check,(&string_array[j][0][0]) + (MAXSTR*i));
                   printf("\nstring i=%d j=%d is %s \n",i,j,check);
                  }
              }
             }
            printf("\nInput of data from cell files to arrays went well\n");
           }
          else
           printf("\nInput of data from cell files to arrays went badly\n");
}

/* function feedback2()                                                    */
/*                                                                         */
/* function displays data arrays from matrix files                         */
/*                                                                         */
/* Input data (passed in as parameters):                                   */
/*  f = eigen spectra matrix                                               */
/*  nc = number of training spectra                                        */
/*  lambda = eigenvalue vectors                                            */
/*  np = L2 retention count                                                */
/*  nloadp = load retention matrix                                         */
/*  beta = regression factors                                              */
/*                                                                         */
/* Output data (displayed to screen):                                      */
/*  f = eigen spectra matrix                                               */
/*  nc = number of training spectra                                        */
/*  lambda = eigenvalue vectors                                            */
/*  np = L2 retention count                                                */
/*  nloadp = load retention matrix                                         */
/*  beta = regression factors                                              */
/*                                                                         */
/* Functions called: none                                                  */
/*                                                                         */
/*************************************************************************/

feedback2(f,nc,lambda,np,nloadp,beta)
float f[][MAXSPEC][MAXTRAIN];/*eigen spectra matrix*/
int nc[];/*number of training spectra*/
float lambda[][MAXTRAIN];/*eigenvalue vectors*/
int np[][MAXCOMPS];/*L2 retention count*/
int nloadp[][NCP1][MAXCOMPS];/*load retention matrix*/
float beta[][NCP1][MAXCOMPS];/*regression factors*/

{
        int i,j,k;/*loop counters*/

        for (i=0;i<5;i++)
         {
          printf("\nCell %d\n",i);

          printf("\nNumber of training spectra is %d\n",nc[i]);

          printf("\nEigenspectra matrix\n");
          for (j=0;j<MAXSPEC;j++)
           {
            printf("\nRow %d\n",j);
            for (k=0;k<MAXTRAIN;k++)
             printf("\nf[%d][%d][%d] = %f\n",i,j,k,f[i][j][k]);
           }
```

```
            printf("\neigenvalue vector\n");
            for (j=0;j<MAXTRAIN;j++)
             printf("\nlambda[%d][%d] = %f\n",i,j,lambda[i][j]);

            printf("\nnp vector\n");
            for (j=0;j<MAXCOMPS;j++)
             printf("\nnp[%d][%d] = %d\n",i,j,np[i][j]);

            printf("\nnloadp matrix\n");
            for (j=0;j<NCP1;j++)
             {
              printf("\nRow %d\n",j);
              for (k=0;k<MAXCOMPS;k++)
               printf("\nnloadp[%d][%d][%d] = %d\n",i,j,k,nloadp[i][j][k]);
             }

            printf("\nBeta matrix\n");
            for (j=0;j<NCP1;j++)
             {
              printf("\nRow %d\n",j);
              for (k=0;k<MAXCOMPS;k++)
               printf("\nbeta[%d][%d][%d] = %f\n",i,j,k,beta[i][j][k]);
             }

           }
}

/* function set_one()                                                          */
/*                                                                             */
/* function sends strings to the selected positioner to                        */
/*     activate the home limit switches                                        */
/*     move all three axes home                                                */
/*     set default velocities and accelerations                                */
/*     set to absolute mode                                                    */
/*                                                                             */
/* input data:                                                                 */
/*  address - furnished by calling function                                    */
/*                                                                             */
/* output data:                                                                */
/*  gofrag - passed to calling function                                        */
/*                                                                             */
/* functions called:                                                           */
/*  writecmd() - send a string to the positioner                               */
/*                                                                             */
/*****************************************************************************/

set_one(address,gofrag)
int address;/*I/O address of PC23 board*/
int *gofrag;/*TRUE => all 3 axes found home*/

{
        int i;/*loop counter*/
        static char message[18][8] /*messages to positioner*/
                = {
        "1LDO ", /*activate axis 1 limit switches*/
        "2LDO ", /*activate axis 2 limit switches*/
        "3LDO ", /*activate axis 3 limit switches*/
        "1GH+5 ", /*move axis 1 to home position*/
        "2GH+5 ", /*move axis 2 to home position*/
        "3GH+5 ", /*move axis 3 to home position*/
        "1RC ", /*did axis 1 go home? */
        "2RC ", /*did axis 2 go home? */
        "3RC ", /*did axis 3 go home? */
        "1A10 ", /*set axis 1 acceleration to 10 rev/sec/sec */
        "2A10 ", /*set axis 2 acceleration to 10 rev/sec/sec */
```

```
        "3A10 ", /*set axis 3 acceleration to 10 rev/sec/sec */
        "1V10 ", /*set axis 1 velocity to 10 rev/sec */
        "2V10 ", /*set axis 2 velocity to 10 rev/sec */
        "3V10 ", /*set axis 3 velocity to 10 rev/sec */
        "1MPA ", /*set axis 1 to absolute mode */
        "2MPA ", /*set axis 2 to absolute mode */
        "3MPA "  /*set axis 3 to absolute mode */
                };
        char s[81];/*status message from positioner*/
        char motormove[8];/*string sent to positioner*/
        int homeflag = TRUE;/*TRUE => axis went home */

        *gofrag = TRUE;
        for (i=0;i<18;i++)
         {
           strcpy(motormove,(&message[0][0])+(8*i));
           printf("\n");
           printf("String %s sent to positioner\n",motormove);
           writecmd(address,motormove);
           if ((i>5) && (i<9))
              {
             readanswer(address,s);
             printf("\nPositioner response is %s\n",s);
             if ((s[2] == '*') && (s[3] == '@'))
              printf("\nPositioner went home\n");
             else
              {
               printf("\nPositioner did not go home\n");
               *gofrag = FALSE;
              }
            }
         }
}

/* function initialize()                                                    */
/*                                                                          */
/* Function to reset the PC23 board                                         */
/* This function assures that the board is ready to accept commands from    */
/* program. The PC23 must have its address set for the first value in the   */
/* user data file, or the program will exit. (ie goflag will clear)         */
/*                                                                          */
/* input data:                                                              */
/*  adr - I/O address of port                                               */
/*                                                                          */
/* output data:                                                             */
/*  gofrag - TRUE => port set up ok                                         */
/*                                                                          */
/* functions called: none                                                   */
/*                                                                          */
/**************************************************************************/

initialize(adr,gofrag)
int adr; /*address of PC23 board*/
int *gofrag;/*TRUE => board set up OK */
{
        /* local declarations and initializations*/
        int i=0;/*loop counter*/

        /*function starts here*/

        /* Page 30 step 1 */
        /* STOP mask forces bits 2, 5 and 6 high*/
        /* bit 2 times out and stops the watchdog timer */
        /* bit 5 restarts the watchdog timer */
        /* bit 6 resets the interrupt latch */
```

```
        outp(adr+1,STOP); /* stop whatever the port is doing */

        /* Page 30 step 2 */
        /* wait for bit 5 to go high  indicates port fail*/
        while (!((inp(adr+1)) & FAIL))
                ;

        /* Page 30 step 3 */
        /* START mask forces bit 6 high */
        /* this resets the interrupt latch */
        outp(adr+1,START);

        /* Page 30 step 4 */
        /* CB mask forces bits 5 and 6 high */
        /* this resets interrupt latch and watchdog timer */
        outp(adr+1,CB);

        /* Page 30 step 5 */
        /* START_M mask is all bits high but 7 */
        /* RESTART mask is bits 0, 1, 2 and 4  high */
        /* indicates motors not running and input data buffer ready */
        /* loop causes the system to wait for this pattern */
        /* to appear, but it gives up after 32000 iterations */
        while (((inp(adr+1) & START_M) != RESTART) && (i++ != MAXINT))
                ;

        if (i >= MAXINT) /* PC23 timed out */
         {
                printf("Initialization of PC23 timed out\n");
                *gofrag = FALSE;/* will force main() to quit */
         }
        else
         {
                *gofrag = TRUE; /*all is well - PC23 did initialize*/

                /* Page 30 step 6 */
                /* INTCLR mask is but 5 high */
                /* restarts watchdog timer */
                outp(adr+1,INTCLR);

                /* Page 30 step 7 */
                /* CB mask forces bits 5 and 6 high */
                /* this resets interrupt latch and watchdog timer */
                outp(adr+1,CB);
         }
}

/* writecmd()                                                              */
/*                                                                         */
/* function writes a command string to the PC23 port                       */
/* program writes command to port by sending one character at a time       */
/*                                                                         */
/* input data:                                                             */
/*  adr - address of port                                                  */
/*  msg - command message                                                  */
/*                                                                         */
/* output data:                                                            */
/*  s - individual message characters sent to port                         */
/*                                                                         */
/* functions called:                                                       */
/*  writech() - send one character to port                                 */
/*                                                                         */
/*************************************************************************/

writecmd(adr,msg)
```

```
int adr;/*address of PC23 board*/
char msg[];/*command string*/
{
        /*this loop starts by pointing at the initial character in */
        /*the array. it writes the character to the address then advances*/
        /*the string pointer. when the thing pointed at by the string */
        /*pointer is 0, as at the end of string marker, the loop quits */
        while (*msg) /*while thing pointed at is not zero, keep going*/
                writech(adr,*msg++); /*write character being pointed at*/
}

/* function readanswer()                                                    */
/*                                                                          */
/* function reads status request response string from PC23                  */
/*                                                                          */
/* input data:                                                              */
/*  addr - address supplied by calling function                             */
/*  msg - string received from PC23 port                                    */
/*                                                                          */
/* output data:                                                             */
/*  msg - string passed back to calling function                            */
/*                                                                          */
/* functions called:                                                        */
/*  readch - reads one character from port                                  */
/*                                                                          */
/**************************************************************************/

readanswer(adr,msg)
int adr;/* address of PC23 port */
char msg[];/* holder for string */
{
        /*go get a character from the port*/
        /*write it to the address pointed at by msg*/
        /*advance pointer to next address*/
        /*keep going until you get end of line marker*/
        while ((*msg++ = readch(adr)) != RETURN)
                ;
        *msg = '\0';/*end of string*/
}


/* Third Level - Functions First Called by Second Level Functions       */


/* function writech()                                                       */
/*                                                                          */
/* Function writes a single character to the PC23 port                      */
/*                                                                          */
/* Input data:                                                              */
/*  addr - address of PC23 port                                             */
/*  alpha - character passed in from calling function                       */
/*                                                                          */
/* Output data:                                                             */
/*  alpha - character written to port                                       */
/*                                                                          */
/* Functions called: None                                                   */
/*                                                                          */
/**************************************************************************/

writech(addr,alpha)
int addr;/*address of PC23 port*/
char alpha;/*character being sent*/
/* Page 30 step 1 - ASCII character passed as parameter */

{
```

```
        /* Page 30 step 2 */
        /*wait until bit 4 becomes 1*/
        /* wait for input data buffer to become ready to receive data*/
        while(!(inp(addr+1) & IDB_M))
                ;

        /* Page 30 step 3 */
        outp(addr,alpha);/*send character*/

        /* Page 30 step 4 */
        /*raise bits 4 5 6 */
        /* bit 4 transmit character from buffer to system*/
        /* bit 5 reset watchdog timer */
        /* bit 6 reset interrupt latch */
        outp(addr+1,IDB);/*send command byte*/

        /* Page 30 step 5 */
        /*wait for input data buffer to clear*/
        while(inp(addr+1) & IDB_M)
                ;
        /* Page 30 step 6 */
        /*reset watchdog timer*/
        /*reset interrupt latch*/
        outp(addr+1,CB);

        /* This step not described on page 30 */
        /*wait for input data buffer to become ready to receive data*/
        while(!(inp(addr+1) & IDB_M))
                ;
}

/*function readch()                                                    */
/*                                                                     */
/* Function reads one character of a PC23 response to a status request */
/* returns character response.                                         */
/*                                                                     */
/* Input data:                                                         */
/*  addr - address provided by calling function                        */
/*  alpha - character provided by PC23                                 */
/*                                                                     */
/* Output data:                                                        */
/*  alpha - character returned to calling function                     */
/*                                                                     */
/* Functions called: None                                              */
/*                                                                     */
/***********************************************************************/

readch(addr)
int addr;/*address of PC23 port*/
{
        /* Page 30 step 1 */
        char alpha=0;/*character to be fetched from PC23 and returned*/

        /* Page 30 step 2 */
        /*wait for a 1 at bit 3 */
        /*wait until output data buffer is ready*/
        while(!(inp(addr+1) & ODB))
                ;

        /* Page 30 step 3 */
        alpha = inp(addr);/*fetch a character from output data buffer*/

        /* Page 30 step 4 */
        /* bit 7 I got it */
        /* bit 6 reset interrupt latch */
```

```
        /* bit 5 reset watchdog timer */
        outp(addr+1,ACK);/*acknowledge receipt of character*/

        /* Page 30 step 5 */
        /*wait until output data buffer is not ready */
        while(inp(addr+1) & ODB)
                ;

        /* Page 30 step 6 */
        /*reset interrupt latch and watchdog timer*/
        outp(addr+1,CB);
        return(alpha);
}
```

```
      program chemo
$NOTRUNCATE
c
c UT Measurement and Control Engineering Center          June 13,1990
c
c M. J. Roberts and S. W. Kercel
c
c chemo.for
c
c Program querys cell description, spectrum description, and spectrum
c vector files; generates chemometric calibration data; writes
c calibration data to a file.
c
c Intended Use:
c The control program for the on-line Raman analyzer requires a
c calibration file for every active cell with two or more components
c in its mixture. Each calibration file is generated by a run of
c this program.
c
c Origins:
c The chemometrics training and verification algorithms were first
c implemented in Fortran in the MacIntosh environment by M. J. Roberts.
c The external matrix manipulation subroutines, rs, which finds the
c eigenvalues and eigenvectors for a real symmetric matrix of general
c size, and matinv, which inverts a matrix of general size, are
c implementations of commercial subroutines. The original program
c was modified to run under Microsoft Fortran in the IBM-PC environment
c and further modified to interface with the on-line Raman analyzer
c control system by S. W. Kercel.
c
c WARNINGS:
c
c This program will overwrite the existing calibration file for
c the cell named in the spectrum description file. Don't run it
c unless you mean to run it, and don't run it until after you've
c backed up any existing calibration files (matrix0.dat through
c matrix4.dat).
c
c This program uses adjustable array bounds. They are TRICKY.
c If you want to modify this program, be aware that a multidimensional
c array declared one size in the main program, and a smaller size in
c a subroutine will use the same memory space, but will organize it
c differently. For example, D(2,2) in the main program points to a
c different place (and holds a different value) than D(2,2) in a
c subroutine. Be forewarned.
c
c This program is compiled using Version 4.1 of the Microsoft Fortran
c Optimizing Compiler. It requires the large memory model. The object
c code must be linked with rs.obj and matinv.obj to resolve external
c references. Attempting to compile this source code with any compiler
c other than the Microsoft Fortran Optimizing Compiler Version 4.1 or
c higher, will lead to unpredictable results.
c
c The spectrum description file, cell description files, spectrum
c vector files, the calibration matrix files, and the three executable
c program codes, spectrum.exe (which generates the spectrum vector
c files), chemo.exe (which uses the spectrum vectors to generate the
c calibration matrix file), and cycle1.exe (which uses the calibration
c matrix file and the current spectrum to generate the current
c composition) are all assumed to reside in the same subdirectory.
c
c As originally written, the fixed array bounds in the main program
c are consistent with the declared constants used in the spectrum and
c cycle programs. That is:
c  MAXSPEC 25 - maximum number of spectrum points per spectrum
```

```
c  MAXCOMPS 4 - maximum number of components in a cell
c  MAXSTR 30 - maximum length of a string describing a component
c  MAXSFILES 30 - maximum sum of count of training and verification
c                 spectrum files for a given cell
c If these constants are changed in the other two programs, then the
c array sizes in the opening declarations and the range checks in
c the getvars subroutine need to be changed accordingly. Since Fortran
c does not support declared constants in the same manner as C, it is
c the responsibility of the programmer to find all the places that need
c to be changed. Sorry about that.
c
c Even with the small limits listed above, the executable code is big.
c If you increase the array sizes significantly, for example, using
c hundreds of spectral points as CIRCOM does, you will probably need to
c resort to heroic measures such as overlays, or running with megabytes
c of memory under OS/2.
c
c Input data:
c  files:
c  spec0.dat - spectrum description file -- contains cell number
c              count of spectra, and for each spectrum,
c              the filename, training/verification flag,
c              and the composition vector
c  cellax.dat - x corresponds to cell number - cell description file
c               contains much useful data, the parts relevant to this
c               program are the active flag, the count of spectrum
c               points, the count of components in the mixture, and
c               the composition strings
c  xxxxxx.dat - spectrum data files -- files listed in spec0.dat
c               contain list of floating point numbers that are the
c               spectral intensities as a function of wave number
c               for the spectra
c
c Output data:
c  file:
c  matrixy.dat - y corresponds to cell number. File contains the data
c                required by the control system to deduce the
c                composition of an unknown mixture, given its spectrum.
c
c Theoretical Basis:
c This program takes the spectra from a set of 'training' runs covering a
c range of properties (in our case chemical concentrations) of the chemical
c constituents and finds a model to estimate the properties of
c an unknown sample from its spectrum.  The technique is that presented in
c the paper, 'Materials Characterization Using Factor Analysis of FT-IR
c Spectra.  Part 2: Mathematical and Statistical Considerations" by
c Fredericks, Lee, Osborn and Swinkels, Applied Spectroscopy, Volume 39,
c Number 2, 1985, page 311.  Where possible I have retained Fredericks'
c nomenclature for variables names.  The "spectra" used here are not full spectra
c for the complete wavenumber range of the instrument but only certain regions
c at which the "action" seems to be happening.
c
c The principal variables used in the analysis are:
c
c nprop   The number of "properties" for which to find models (<=4).
c         A property, in our case, is a chemical concentration.
c
c pname   A vector of names of the properties for later reference in the
c         report of results.  It has nprop elements.
c
c nc      The number of different concentration combinations (mixtures) used
c         in the training set (<=20).
c
c prop    A matrix containing the "property" information.  Each row of
c         prop contains the numerical values for a single property.  Its
```

```
c         size is nprop rows by nc columns.
c
c nr      The number of wavenumbers used from each spectrum of
c         Raman responses for training (<=25).
c
c D       A matrix of data representing the spectra for the set of
c         chemical concentrations used in training.  Each column
c         of the matrix is a spectrum for a particular mixture.
c         Its size is nr rows by nc columns.
c
c Z       The "product-moment" or "dispersion" matrix of Fredericks,
c         et. al.  It is the product of D transpose with D and is
c         therefore square.  Its size is nc rows by nc columns.
c
c lambda  A vector of the eigenvalues of Z with nc elements.
c
c L       A matrix whose columns are the eigenvectors of Z.  It is
c         the "loading" matrix of Fredericks, et. al.  It is
c         square.  Its size is nc rows by nc columns.
c
c F       A matrix whose columns are the abstract "factors" or
c         "eigenspectra" of D.  It is the same size as D.
c
c L2      The augmented loading matrix.  Augmented by addition of
c         a column of 1's to allow for a constant offset in the
c         multiple linear regression (MLR) model.  Its size is nc
c         rows by (nc+1) columns.
c
c re
c ie
c xe
c IND     Four vectors containing the "empirical error" data
c         described in the paper "Determination of the Number
c         of Factors and the Experimental Error in a Data
c         Matrix" by Edmund R. Malinowski, Analytical Chemistry,
c         Vol. 49, Number 4, April 1977, page 612.  These are used
c         to reduce the number of loadings used.  Each vector has
c         nc elements but only the first (nc-1) are significant.
c
c nf      The number of loading matrix columns of matrix L retained
c         after the empirical tests.  (The number of columns retained
c         in the augmented loading matrix L2 is (nf+1).)
c
c B2prop  A matrix of the coefficients in the regression models used
c         to predict properties.  Each column contains the set of
c         coefficients for one property.  Its size is nc rows by nprop
c         columns.
c
c B2punc  A matrix of the uncertainties of the coefficients in the
c         models.  Each column contains the uncertainties in the
c         coefficients for one property.  Its size is nc rows by nprop
c         columns.
c
c pest    A matrix containing the estimates of the properties.  Each
c         column contains the estimates of a single property corresponding
c         to one of the mixtures.  Its size is nc rows by nprop columns.
c
c Meanp   A vector of the mean values of the properties.  It has nprop
c         elements.
c
c Varp    A vector of the variances of the values of the properties.
c         It has nprop elements.
c
c MeanL   A vector of mean values of the columns of L.  It has nc elements.
c
```

```
c VarL    A vector of variances of the columns of L.  It has nc elements.
c
c rhopL   A matrix of correlation coefficients between properties
c         and the columns of L.  Each column contains the correlation
c         coefficients between a property and the columns of L.  Its
c         size is nc rows by nprop columns.
c
c rhothr  A vector of "threshold" correlation coefficients, one for
c         each property.  Factor loadings for which the correlation
c         coefficient exceeds this threshold are retained; the others
c         are droppe.  This vector has nprop elements.
c
c unc     The number of different concentration combinations in the set
c         of unknown samples.
c
c uprop   A matrix containing the numerical properties of the unknowns.
c         Each row of uprop contains the numerical data representing
c         one property.  Its size is nprop rows by nc columns.
c
c uD      The "D" matrix of spectra of the "unknown" solutions.  Its
c         size is nr rows by unc columns.
c
c uL2     The loading matrix for the unknowns, unc by nc+1.
c
c upest   A matrix containing the estimates of the unknown properties.  Each
c         column contains the estimates of a single property corresponding
c         to one of the mixtures.  Its size is unc rows by nprop columns.
c
c np      A vector of integers.  Each element is the number of columns of
c         the augmented loading matrix, L2, which are kept after examining
c         the correlation between as property and the loading matrix columns.
c         It has nprop elements.
c
c L3      The modified, augmented loading matrix.  It is modified by
c         retaining only those loading factors with significant correlations
c         to a property.  Thus the factors retained in any particular
c         analysis depend on the property in question.  The matrix is
c         declared nc by (nc+1) by nprop but the number of columns in each
c         rank used in the MLR process will be np which varies from
c         property to property and is less than or equal to (nf+1).
c
c B3prop  A reduced matrix of the coefficients in the regression models used
c         to predict properties.  Each column contains the set of
c         coefficients for one property.  Its size is nc rows by nprop
c         columns but only the first np rows in each column are signficant.
c
c B3punc  A reduced matrix of the uncertainties of the coefficients in the
c         models.  Each column contains the uncertainties in the
c         coefficients for one property.  Its size is nc rows by nprop
c         columns but only the first np rows in each column are significant.
c
c nloadp  A matrix of integers.  Each column contains the subscripts
c         designating the loading columns of L2 which are retained
c         for a property to form L3.  Its size is (nc+1) rows by nprop
c         columns but only the first np (for a particular property)
c         elements in a particular column are significant.
c
c uL3p    This is a three-dimensional matrix.  Each rank corresponds to
c         a property.  Within each rank each column contains the loadings
c         for a factor and each row corresponds to one of the unknown
c         mixtures. Its size is unc rows by (nc+1) columns by nprop
c         ranks.
c
c n
c n
```

```
c ierr
c matz    are used by the subroutine "rs" and their use
c         is explained there
c
c-----------------------------------------------------------------------
c main program starts here
c-----------------------------------------------------------------------
c
        integer n,nm,ierr,matz,rows,cell
        integer nprop,nc,nr,unc,ncp1,nf,nfp1,np(4)
        integer nloadp(21,4)
        real*4 prop(20,4),pest(20,4)
        real*4 uprop(20,4),upest(20,4)
        real*8 Meanp(4),Varp(4)
        real*8 MeanL(20),VarL(20)
        real*8 rhopL(20,4),rhothr(4)
        real*8 Z(20,20),lambda(20),L(20,20),fv1(20),fv2(20)
        real*8 D(25,20),uD(25,20),F(25,20)
        real*8 L2(20,21),uL2(3,21),L3(20,21,4)
        real*8 uL3p(20,21,4)
        real*8 B2prop(21,4),B2punc(21,4)
        real*8 B3prop(21,4),B3punc(21,4)
        real*8 LTL(20,20),re(20),ie(20),xe(20),IND(20)
        real*8 sprop(4)
        character*30 pname(4)
        character*13 filen
        character*1 diag
        character*13 tfiles(20),vfiles(20)
        real*4 dprop(4)
        logical goflag

c
c Prompt the user to decide on diagnostics display.
c
        write(*,100)
100     format(' Do you want full diagnostics? (Y/N)')
        read(*,101) diag
101     format(a1)
c
c Get the training training properties and the corresponding
c 'D' matrix from disk.  Its size is nr rows by nc columns.  Each column
c is a vector of Raman scattering responses at wavelengths chosen for
c analysis and for a particular combination of properties.
c There are therefore nc different combinations of properties used
c in the training set.
c Get the unknown properties and the corresponding 'uD' matrix from disk.
c Its size is nr rows by unc columns.  Each column is a vector of Raman
c scattering responses at wavelengths chosen for analysis and for a
c particular combination of properties.  There are therefore unc
c different combinations of properties used in the unknown set.
c
      call getvars(goflag,nr,nprop,nc,unc,cell)
      if (.not. goflag) stop
      call getarrs(nprop,nc,nr,unc,pname,prop,uprop,tfiles,vfiles,
     1dprop,D,uD)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          write(*,2002)
2002      format(' For the training set')
          call reptD(nprop,nr,nc,pname,prop,D)
          write(*,2004)
2004      format(' For the unknown set')
          call reptD(nprop,nr,unc,pname,uprop,uD)
        endif
c
c Form the product-moment matrix, Z.  It is real, symmetric and nc by nc.
```

```
c
        call formZ(nr,nc,D,Z)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          write(*,1002)
1002      format(' The Z matrix is')
          call repmd(nc,nc,Z)
        endif
        nm=nc
        n=nc
        matz=1
c
c The usage of nm, n, and matz is described in the comments on the subroutine
c 'rs'.  Subroutine rs calculates the eigenvalues and eigenvectors of matrix, Z.
c
        write(*,1003)
1003    format(' Finding eigenvalues and eigenvectors of Z...'/)
        call rs(nm,n,Z,lambda,matz,L,fv1,fv2,ierr)
c
c Arrange the eigenvalues (and corresponding eigenvectors) in descending
c order.
c
        call descnd(nc,lambda,L)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          write(*,1004)
1004      format(' The eigenvalues are')
          call repvd(nc,lambda)
          write(*,1005)
1005      format(' The eigenvectors are')
          call repmd(nc,nc,L)
          call chkev(nr,nc,L,Z,lambda,D,LTL)
        endif
c
c Form the abstract factor or 'eigenspectrum' matrix, F.
c
        call formF(nr,nc,D,L,F)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          write(*,1006)
1006      format(' The F matrix (up to 1st 40 rows) is')
          rows=nr
          if (rows.gt.40) rows=40
          call repmd(rows,nc,F)
        endif
c
c Do some empirical tests on the data to possibly eliminate the noisy
c eigenvalues and eigenspectra from the model.  These tests are described
c in the paper, 'Determination of the Number of Factors and the Experimental
c Error in a Data Matrix' by Edmund R. Malinowski, Analytical Chemistry,
c Volume 49, Number 4, April 1977.  'nf' is the number of eigenvalues
c chosen to be retained in estimating properties after the tests are made.
c
        call tests(nr,nc,lambda,re,ie,xe,IND,nf)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          call reptst(nc,nf,re,ie,xe,IND)
        endif
c
c Form the augmented matrix, L2.
c
        ncp1=nc+1
        call formL2(nc,ncp1,L,L2)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          write(*,1007)
1007      format(' The L2 matrix is')
          call repmd(nc,ncp1,L2)
        endif
c
```

```
c Form the matrix of 'betas' which are the coefficients of the
c loading factors in the MLR model and display the estimates
c of the properties versus the actual properties.
c
        call MLR2(nprop,nc,ncp1,nf,pname,prop,L2,B2prop
     1  ,B2punc,pest,sprop)
c
c Form the loading matrix for the unknowns.
c
        call frmuL2(nr,unc,nc,ncp1,uD,F,lambda,uL2)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          write(*,1008)
1008      format(' The uL2 matrix is')
          call repmd(unc,ncp1,uL2)
        endif
        write(*,30)
30      format(' Estimating unknown properties...'/)
        call estup2(nprop,unc,ncp1,nf,pname,uprop,B2prop,uL2,upest)
c
c Find the means and variances of the properties.
c
        call propst(nprop,nc,prop,Meanp,Varp)
        do 99999 iprop=1,nprop
          if ((diag.eq.'Y').or.(diag.eq.'y')) then
            write(*,1009) pname(iprop),Meanp(iprop)
     1          ,pname(iprop),Varp(iprop)
1009        format(' Mean of ',a,' is',0PF8.3,', Variance of '
     1      ,a,'is ',0PF8.3)
          endif
99999   continue
c
c Find the means and variances of the columns of the loading matrix, L.
c
        call Lstat(nc,nf,L,MeanL,VarL)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          write(*,1012)
1012      format(' The means of the columns of the L matrix are')
          call repvd(nc,MeanL)
          write(*,1013)
1013      format(' The variances of the columns of the L matrix are')
          call repvd(nc,VarL)
        endif
c
c Calculate and display the correlation of the properties
c with the loading factors for the number of factors
c retained (so far), nf.
c
        write(*,40)
40      format(' Correlating properties with loadings...'/)
        call corr(nprop,nc,nf,pname,prop,Meanp,Varp,L,MeanL
     1  ,VarL,rhopL)
        do 99998 iprop=1,nprop
          rhothr(iprop)=0.1
99998   continue
        if ((diag .eq. 'Y') .or. (diag .eq. 'y')) then
          write(*,1014) nf
1014      format(' Correlation coef. matrix'/
     1    '  (Only the first ',i2,' rows are significant)')
          call repmd(nc,nprop,rhopL)
        endif
        call MLR3(nprop,nc,ncp1,nf,pname,prop,L2,rhopL,rhothr
     1  ,B3prop,B3punc,pest,np,nloadp,L3,sprop)
c
c Form the new loading matrices for the unknowns.
c
```

```
        call frmuL3(nprop,unc,ncp1,np,uL2,nloadp,uL3p)
        if ((diag.eq.'Y').or.(diag.eq.'y')) then
          write(*,1017)
1017      format(' The uL3 matrix is')
c          call repmd3(unc,ncp1,uL3p)
        endif
        write(*,33)
33      format(' Estimating properties of the unknowns...'/)
        call estup3(nprop,unc,ncp1,np,pname,uprop,B3prop
     1  ,uL3p,upest)
        call writemat(cell,nprop,nr,ncp1,nc,f,lambda,np,nloadp,B3prop)
        stop
        end
c
c-----------------------------------------------------------------------
c Subroutines start here
c-----------------------------------------------------------------------
c
c
c The validity of the eigenvalues is checked in two ways.  First by
c forming the sum of the squares of all the data points in D and the sum
c of all the eigenvalues.  They should be equal.  Second by forming two
c vectors; the product of Z with an eigenvector and the product of the
c eigenvalue corresponding to that eigenvector with that eigenvector.
c The resultant vectors should be the same.  This second procedure is done
c for all eigenvalues and eigenvectors.
c
c This subroutine is for diagnosis only and would not normally be in
c the final version.
c
        subroutine chkev(nr,nc,L,Z,lambda,D,LTL)
        integer nr,nc
        real*8 L(nc,nc),LTL(nc,nc),Z(nc,nc),lambda(nc)
        real*8 D(nr,nc),vec1,vec2
        do 99997 i=1,nc
          do 99998 j=1,nc
            LTL(i,j)=0
            do 99999 k=1,nc
              LTL(i,j)=LTL(i,j)+L(k,i)*L(k,j)
99999       continue
99998     continue
99997   continue
        write(*,5)
5       format(' L transpose L:')
        do 99996 i=1,nc
          write(*,10) (LTL(i,j),j=1,nc)
10        format(16(0PF6.2))
99996   continue
        pause 'Hit return to continue'
        sumDsq=0
        do 99994 i=1,nr
          do 99995 j=1,nc
            sumDsq=sumDsq+D(i,j)**2
99995     continue
99994   continue
        sumlam=0
        do 99993 i=1,nc
          sumlam=sumlam+lambda(i)
99993   continue
        write(*,15) sumDsq,sumlam
15      format(' The sum of the squares of the data points in D is '
     1  ,1pe15.7/' The sum of the eigenvalues is ',1pe15.7)
c do (i=1,nc)
c   write(*,20) i
c20       format(' For eigenvalue #',i2,', Z*l and lambda*l are:')
```

```
c   do (j=1,nc)
c     vec1=0
c     do (k=1,nc)
c       vec1=vec1+Z(j,k)*L(k,i)
c     continue
c     vec2=lambda(i)*L(j,i)
c     write(*,30) vec1,vec2
c30         format(1pe23.15,1pe23.15)
c   continue
c   pause 'Hit return to continue'
c continue
        return
        end

c
c Subroutine correlate finds the correlation coefficients between
c each property and the loadings in the columns of 'L'.
c
        subroutine corr(nprop,nc,nf,pname,prop,Meanp
     1  ,Varp,L,MeanL,VarL,rhopL)
        integer nprop,nc,nf
        real*4 prop(nc,nprop)
        real*8 Meanp(nprop),MeanL(nc),Varp(nprop),VarL(nc)
        real*8 L(nc,nc),sumpL,CovpL,rhopL(nc,nprop)
        character*30 pname(nprop)
c
c Cov(x,y)=E(xy)-E(x)E(y)
c Correlation coefficient, rho = Cov(x,y)/Sqrt(Var(x)*Var(y))
c
        do 99997 iprop=1,nprop
          do 99998 j=1,nf
            sumpL=0
            do 99999 i=1,nc
              sumpL=sumpL+L(i,j)*prop(i,iprop)
99999       continue
            CovpL=sumpL/nc-Meanp(iprop)*MeanL(j)
            rhopL(j,iprop)=CovpL/dsqrt(Varp(iprop)*VarL(j))
c     write(*,10) j,rhopL(j,iprop)
c10         format(' Factor #',i2,', Corr. Coef. = ',0pf12.7)
99998     continue
99997   continue
        return
        end

c
c
c Subroutine descend rearranges the eigenvalues and corresponding
c eigenvectors in descending order of eigenvalue.  This is not a
c bubble sort but a deterministic re-ordering because the eigenvalues
c as reported by 'rs' are in ascending order.  Therefore this subroutine
c simply reverses the order.
c
        subroutine descnd(nc,lambda,L)
        integer nc
        real*8 lambda(nc),L(nc,nc),temp
        write(*,1)
1       format(' Arranging eigenvalues in descending order...'/
     1  ,'  (Also rearranging eigenvectors to correspond)'/)
        nc2=nc/2
        do 99998 j=1,nc2
          jj=nc-j+1
          temp=lambda(j)
          lambda(j)=lambda(jj)
          lambda(jj)=temp
          do 99999 i=1,nc
```

```
            temp=L(i,j)
            L(i,j)=L(i,jj)
            L(i,jj)=temp
99999      continue
99998    continue
         return
         end

c
c Subroutine estup2 estimates the value of a property using the model
c already established and the loadings just computer for the unknown
c spectra.
c
         subroutine estup2(nprop,unc,ncp1,nf,pname,uprop,B2prop
     1   ,uL2,upest)
         integer nprop,unc,nf,nfp1,ncp1
         real*4 uprop(unc,nprop),upest(unc,nprop)
         real*8 B2prop(ncp1,nprop),uL2(unc,ncp1)
         character*30 pname(nprop)
         nfp1=nf+1
         do 99997 iprop=1,nprop
           write(*,10) pname(iprop)
10         format(1x,a)
           do 99998 i=1,unc
             upest(i,iprop)=0
             do 99999 j=1,nfp1
               upest(i,iprop)=upest(i,iprop)+uL2(i,j)*
     1            B2prop(j,iprop)
99999        continue
             percerr=upest(i,iprop)-uprop(i,iprop)
             write(*,20) upest(i,iprop),uprop(i,iprop),percerr
20           format(' Est. of unk. property is',0PF8.3,'
     1        ,  Actual unk. property is',0PF8.3,', Error is ',0PF8.3)
99998      continue
           pause 'Hit return to continue'
99997    continue
         return
         end

c
c Subroutine estup3 estimates the value of a property using the model
c already established and the loadings just computer for the unknown
c spectra.
c
         subroutine estup3(nprop,unc,ncp1,np,pname,uprop
     1   ,B3prop,uL3p,upest)
         integer nprop,unc,ncp1,np(nprop)
         real*4 uprop(unc,nprop),upest(unc,nprop)
         real*8 B3prop(ncp1,nprop),uL3p(unc,ncp1,nprop)
         character*30 pname(nprop)
         do 99997 iprop=1,nprop
           write(*,10) pname(iprop)
10         format(1x,a)
           do 99998 i=1,unc
             upest(i,iprop)=0
             do 99999 j=1,np(iprop)
               upest(i,iprop)=upest(i,iprop)
     1                 +uL3p(i,j,iprop)*B3prop(j,iprop)
99999        continue
             percerr=upest(i,iprop)-uprop(i,iprop)
             write(*,20) upest(i,iprop),uprop(i,iprop),percerr
20           format(' Est. of unk. property is',0PF8.3,'
     1        ,  Actual unk. property is',0PF8.3,'
     2        , Error is ',0PF8.3)
99998      continue
```

```
          pause 'Hit return to continue'
99997   continue
        return
        end

c
c Subroutine formB2 is a generic subroutine to form the matrix, B2prop,
c of coefficients in the MLR model to predict any property.
c Also the uncertainties of the coefficients, B2punc, are calculated.
c
        subroutine formB2(nprop,nc,ncp1,nfp1,L2,L2TL2I,pname
     1  ,prop,B2prop,B2punc,pest,sprop)
        integer nprop,nc,ncp1,nfp1
        real*4 prop(nc,nprop),pest(nc,nprop)
        real*8 L2(nc,ncp1),L2TL2I(21,21),L2Tp(21,4)
        real*8 B2prop(ncp1,nprop),B2punc(ncp1,nprop),sprop(nprop)
        character*30 pname(nprop)
        do 99992 iprop=1,nprop
          write(*,10) pname(iprop)
10        format(a)
          do 99998 i=1,nfp1
            L2Tp(i,iprop)=0
            do 99999 j=1,nc
              L2Tp(i,iprop)=L2Tp(i,iprop)+L2(j,i)*prop(j,iprop)
99999       continue
99998     continue
          do 99996 i=1,nfp1
            B2prop(i,iprop)=0
            do 99997 j=1,nfp1
              B2prop(i,iprop)=B2prop(i,iprop)+
     1          L2TL2I(i,j)*L2Tp(j,iprop)
c       write(*,11) iprop,i,j,B2prop(i,iprop)
c     1          ,L2TL2I(i,j),L2Tp(j,iprop)
c11           format(' iprop=',i3,' i=',i3,' j=',i3,' B2prop='
c     1         ,1pe11.3,' L2TL2I=',1pe11.3,' L2Tp=',1pe11.3)
99997       continue
99996     continue
          rss=0
          do 99994 i=1,nc
            pest(i,iprop)=0
            do 99995 j=1,nfp1
              pest(i,iprop)=pest(i,iprop)+L2(i,j)*B2prop(j,iprop)
99995       continue
            rss=rss+(pest(i,iprop)-prop(i,iprop))**2
            percerr=pest(i,iprop)-prop(i,iprop)
            write(*,20) pest(i,iprop),prop(i,iprop),percerr
20          format(' Est. of property is',0PF8.3,'
     1     ,  Actual property is',0PF8.3,', Error is ',0PF8.3)
99994     continue
          estvar=rss/(nc-nfp1-1)
          sprop(iprop)=sqrt(estvar)
          write(*,30) sprop(iprop)
30        format(' The estimate of the std. dev. of the '
     1    ,'input data noise is ',1pe15.7)
c   pause 'Hit return to continue'
          write(*,40)
40        format(' The coefficients and their uncertainties are:')
          do 99993 i=1,nfp1
            B2punc(i,iprop)=sprop(iprop)*sqrt(L2TL2I(i,i))
            percerr=B2punc(i,iprop)*100/B2prop(i,iprop)
            write(*,50) i,B2prop(i,iprop),B2punc(i,iprop),percerr
50          format(' Coef #',i2,' = ',0PF8.3,', with unc. of '
     1      ,0PF8.3,' or ',0PF8.3,' %')
99993     continue
          pause 'Hit return to continue'
```

```
99992   continue
        return
        end

c
c Subroutine formB3 is a generic subroutine to form the matrix, B3prop,
c of coefficients in the MLR model to predict any property.
c  Also the uncertainties of the coefficients, B3punc, are calculated.
c
        subroutine formB3(col,nprop,nc,ncp1,np,L3,L3TL3I
     1  ,pname,prop,B3prop,B3punc,pest,sprop)
        integer nprop,nc,ncp1,np(nprop),col
        real*4 prop(nc,nprop),pest(nc,nprop)
        real*8 sprop(nprop)
        real*8 L3(nc,ncp1,nprop),L3TL3I(21,21),L3Tp(21)
        real*8 B3prop(ncp1,nprop),B3punc(ncp1,nprop)
        character*30 pname(nprop)
        write(*,1) pname(col)
1       format(a)
        do 99998 i=1,np(col)
          L3Tp(i)=0
          do 99999 j=1,nc
            L3Tp(i)=L3Tp(i)+L3(j,i,col)*prop(j,col)
99999     continue
99998   continue
        do 99996 i=1,np(col)
          B3prop(i,col)=0
          do 99997 j=1,np(col)
            B3prop(i,col)=B3prop(i,col)+L3TL3I(i,j)*L3Tp(j)
99997     continue
99996   continue
        rss=0
        do 99994 i=1,nc
          pest(i,col)=0
          do 99995 j=1,np(col)
            pest(i,col)=pest(i,col)+L3(i,j,col)*B3prop(j,col)
99995     continue
          rss=rss+(pest(i,col)-prop(i,col))**2
          percerr=pest(i,col)-prop(i,col)
          write(*,20) pest(i,col),prop(i,col),percerr
20        format(' Est. of property is',OPF8.3,'
     1     ,  Actual property is',OPF8.3,', Error is ',OPF8.3)
99994   continue
        estvar=rss/(nc-np(col)-1)
        sprop(col)=sqrt(estvar)
        write(*,30) sprop(col)
30      format(' The estimate of the std. dev. of the '
     1  ,'input data noise is ',1pe15.7)
c pause 'Hit return to continue'
        write(*,40)
40      format(' The coefficients and their uncertainties are:')
        do 99993 i=1,np(col)
          B3punc(i,col)=sprop(col)*dsqrt(L3TL3I(i,i))
          percerr=B3punc(i,col)*100/B3prop(i,col)
          write(*,50) i,B3prop(i,col),B3punc(i,col),percerr
50        format(' Coef #',i2,' = ',1pe15.7,', with unc. of '
     1    ,1pe15.7,' or ',1pe15.7,' %')
99993   continue
        return
        end

c
c Subroutine formF forms the matrix, F, of abstract factors or
c 'eigenspectra'.  It is the product of D with L and is therefore
c nr by nc.  Each column of F is an eigenspectrum, the most significant
```

```
c being in the first column, the next most significant in the second
c column, etc...
c
        subroutine formF(nr,nc,D,L,F)
        integer nr,nc
        real*8 D(nr,nc),L(nc,nc),F(nr,nc)
        write(*,1)
1       format(' Forming F matrix of eigenspectra...'/)
        do 99997 i=1,nr
          do 99998 j=1,nc
            F(i,j)=0
            do 99999 k=1,nc
              F(i,j)=F(i,j)+D(i,k)*L(k,j)
99999       continue
99998     continue
99997   continue
        return
        end
c
c Augment the matrix, L, to matrix, L2 by adding a column of
c one's as the first column and shifting all other columns to
c the right.  This allows for a constant offset in the multiple
c linear regression model.
c
        subroutine formL2(nc,ncp1,L,L2)
        integer nc,ncp1
        real*8 L(nc,nc),L2(nc,ncp1)
        write(*,1)
1       format(' Forming augmented loading matrix...'/)
        do 99998 i=1,nc
          L2(i,1)=1
          do 99999 j=2,ncp1
            L2(i,j)=L(i,j-1)
99999     continue
99998   continue
        return
        end
c
c
c Subroutine formZ forms the product-moment or dispersion matrix, Z.
c It is the product of D transpose with D and is therefore nc by nc.
c
        subroutine formZ(nr,nc,D,Z)
        integer nr,nc
        real*8 D(nr,nc),Z(nc,nc)
        write(*,1)
1       format(' Forming product-moment or Z matrix...'/)
        do 99997 i=1,nc
          do 99998 j=1,nc
            Z(i,j)=0
            do 99999 k=1,nr
              Z(i,j)=Z(i,j)+D(k,i)*D(k,j)
99999       continue
99998     continue
99997   continue
        return
        end
c
c Subroutine frmuL2 forms the loading matrix for the unknowns
c by using the information in the eigenspectra, F, and the
c spectra of the unknowns, uD.
c
        subroutine frmuL2(nr,unc,nc,ncp1,uD,F,lambda,uL2)
        integer nr,unc,nc,ncp1
        real*8 uD(nr,unc),F(nr,nc),lambda(nc),uL2(unc,ncp1)
```

```
        write(*,1)
1       format(' Forming loading vectors for unknowns...'/)
        ncp1=nc+1
        do 99997 i=1,unc
          uL2(i,1)=1
          do 99998 j=2,ncp1
          uL2(i,j)=0
            do 99999 k=1,nr
              uL2(i,j)=uL2(i,j)+F(k,j-1)*uD(k,i)
99999       continue
            uL2(i,j)=uL2(i,j)/lambda(j-1)
99998     continue
c   write(*,10) (uL2(i,j),j=1,ncp1)
c10       format(50(1pe11.3))
99997   continue
        return
        end
c
c     Subroutine frmuL3 forms the loading matrix for the unknowns
c     by using the information in the eigenspectra, F, and the
c     spectra of the unknowns, uD.
c
      subroutine frmuL3(nprop,unc,ncp1,np,uL2,nloadp,uL3p)
      integer nprop,unc,ncp1,np(nprop),nloadp(ncp1,nprop)
      real*8 uL2(unc,ncp1),uL3p(unc,ncp1,nprop)
      write(*,1)
1     format(' Forming new loading vectors for unknowns...'/)
c
c     Stack the retained loading factors in the first np columns of uL3.
c
      do 99997 iprop=1,nprop
        do 99998 j=1,np(iprop)
          jj=nloadp(j,iprop)
          do 99999 i=1,unc
           uL3p(i,j,iprop)=uL2(i,jj)
99999     continue
99998   continue
99997 continue
      return
      end

c subroutine getarrs
c
c this subroutine  reads the spectrum description file and the cell
c description file, loads data from the files to various arrays,
c finds the spectrum files and loads the D and uD arrays;
c closes both description files, opens and closes all spectrum files
c
c Input data:
c  files:
c  spec0.dat - spectrum description file -- contains cell number
c              count of spectra, and for each spectrum,
c              the filename, training/verification flag,
c              and the composition vector
c  cellax.dat - x corresponds to cell number - cell description file
c               contains much useful data, the parts relevant to this
c               program are the active flag, the count of spectrum
c               points, the count of components in the mixture, and
c               the composition strings
c  xxxxxx.dat - spectrum data files -- files listed in spec0.dat
c               contain list of floating point numbers that are the
c               spectral intensities as a function of wave number
c               for the spectra
c  parameters passed in from caller:
c  nr - number of rows in D spectrum -- number of points per spectrum
```

```
c  nprop - number of components in the mixture in the cell
c  nc - number of columns in D spectrum -- number of training samples
c  unc - nr of columns in uD spectrum -- nr of verification samples
c
c Output data (parameters passed back to caller):
c  goflag - .true. => all required data successfully read from files
c  pname() - vector of strings of property names
c  prop() - matrix of properties (compositions) for training spectra
c           each row is the property vector for a given sample;
c           each component of the property vector is a composition for
c           the component in the corresponding position in pname()
c           matrix has nprop columns and nc rows
c  uprop() - matrix of properties (compositions) for verification
c            spectra, each row is the property vector for a sample
c            each component of the property vector is a composition for
c            the component in the corresponding position in pname()
c            matrix has nprop columns and unc rows
c  D - training spectrum matrix, each column is a list of intensities
c      there are nr rows and nc columns
c  uD - verification spectrum matrix, each column is a list of
c       intensities; there are nr rows and unc columns
c
c Subprograms called: None
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine getarrs(nprop,nc,nr,unc,pname,prop,uprop,tfiles,
     1vfiles,dprop,D,uD)
      integer nprop,nc,nr,unc
      character*30 pname(nprop)
      real*4 prop(nc,nprop),uprop(unc,nprop)
      character*13 tfiles(nc),vfiles(unc)
      real*4 dprop(nprop)
      real*8 D(nr,nc),uD(nr,unc)
      integer nspecs,tvf,tcount,vcount
      character*13 adummy

c
c read the mixture component names from the cell description file
c
      do 99999 i=1,nprop
99999 read(2,200) pname(i)
200   format(a)
      close(2)
201   format(i3)
202   format(f6.3)
c
c from the spectrum description file, for each spectrum, read the
c spectrum data filename, and the composition vector
c put in the training array or the verification array as flagged
c
      nspecs = nc + unc
      tcount = 0
      vcount = 0
      do 99998 i=1,nspecs
       read(1,200) adummy
       read(1,201) tvf
       do 99991 j=1,nprop
99991  read(1,202) dprop(j)
       if (tvf .eq. 1) then
        tcount = tcount + 1
        tfiles(tcount) = adummy
         do 99997 k=1,nprop
99997    prop(tcount,k) = dprop(k)
       else
        vcount = vcount + 1
```

```
        vfiles(vcount) = adummy
         do 99996 k=1,nprop
99996    uprop(vcount,k) = dprop(k)
       endif
99998 continue
      close(1)
c
c go through the list of training spectrum files. each becomes a column
c in the D array. Left to right sequence of columns in the D array
c matches top to bottom sequence of compositions in prop array
c
203   format(g16.0)
      do 99990 i=1,nc
       adummy = tfiles(i)
       open(1,file=adummy)
       do 99989 j=1,nr
       read(1,203) D(j,i)
99989  continue
       close(1)
99990 continue
c
c go through the list of verification spectrum files. each becomes a column
c in the uD array. Left to right sequence of columns in the uD array
c matches top to bottom sequence of compositions in uprop array
c
      do 99988 i=1,unc
       adummy = vfiles(i)
       open(1,file=adummy)
       do 99987 j=1,nr
99987  read(1,203) uD(j,i)
       close(1)
99988 continue

      return
      end

c subroutine getvars
c
c this subroutine  reads the spectrum description file and the cell
c description file, loads array dimension data to key variables,
c opens both description files
c
c Input data:
c  files:
c  spec0.dat - spectrum description file -- contains cell number
c              count of spectra, and for each spectrum,
c              the filename, training/verification flag,
c              and the composition vector
c  cellax.dat - x corresponds to cell number - cell description file
c               contains much useful data, the parts relevant to this
c               program are the active flag, the count of spectrum
c               points, the count of components in the mixture, and
c               the composition strings
c
c Output data:
c  parameters passed back to caller:
c  goflag - .true. => all required data successfully read from files
c  nr - number of rows in D spectrum -- number of points per spectrum
c  nprop - number of components in the mixture in the cell
c  nc - number of columns in D spectrum -- number of training samples
c  unc - nr of columns in uD spectrum -- nr of verification samples
c  cell - number of cell being calibrated
c
c Subprograms called: None
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
      subroutine getvars(goflag,nr,nprop,nc,unc,cell)
      logical goflag
      integer nr,nprop,nc,unc
      integer cell,active,dummy,nspecs
      character*13 cellfile
      real*4 rdummy
      character*1 adummy

c
c First: query the spectrum description file to find out which
c sample cell is to be calibrated
c
      goflag = .true.
      open(1,file='spec0.dat')
      read(1,30) cell
30    format(i3)
      if ((cell .lt. 0) .or. (cell .gt. 4)) then
       goflag = .false.
       write(*,31)
31     format(' Cell number out of range 0-4')
       return
      endif
      if (cell .eq. 0) cellfile = 'cella0.dat'
      if (cell .eq. 1) cellfile = 'cella1.dat'
      if (cell .eq. 2) cellfile = 'cella2.dat'
      if (cell .eq. 3) cellfile = 'cella3.dat'
      if (cell .eq. 4) cellfile = 'cella4.dat'
c
c Next query the cell description file to find out if the cell is active
c
      open(2,file=cellfile)
      read(2,30) active
      if (active .ne. 1) then
       goflag = .false.
       write(*,32)
32     format(' Cell not active')
       return
      endif
c
c positioner coordinates, number of scans, and normalization flag have
c no meaning in this module, move the file pointer past these data
c
      read(2,30) dummy
      read(2,30) dummy
      read(2,30) dummy
      read(2,33) rdummy
33    format(f6.3)
34    format(i1)
35    format(a1)
36    format(i2)
c
c find the number of spectral points per spectrum
c
      read(2,30) nr
      if ((nr .lt. 0) .or. (nr .gt. 25)) then
       goflag = .false.
       write(*,37)
37     format(' Number of spectral components out of range 0-25')
       return
      endif
c
c move file pointer past the spectral wavelengths
c
      do 99999 i=1,nr
```

```
99999 read(2,30) dummy
c
c find out how many components the mixture in the cell has
c
      read(2,30) nprop
      if ((nprop .lt. 2) .or. (nprop .gt. 4)) then
       goflag = .false.
       write(*,38)
38     format(' Number of properties out of range 2-4')
       return
      endif
c
c find out how many spectra are described in the spectrum description
c
      read(1,30) nspecs
c
c loop through the spectrum list and count the number of spectra
c used for training (flag = 1) and for verification (flag = 0)
c
      nc=0
      unc=0
      do 99998 i=1,nspecs
       read(1,35) adummy
       read(1,34) dummy
       if (dummy .eq. 1) then
          nc = nc + 1
       else
        unc = unc + 1
       endif
      do 99996 j=1,nprop
99996 read(1,34) dummy
99998 continue
      if ((nc .lt. 0) .or. (nc .gt. 20)) then
       goflag = .false.
       write(*,39)
39     format(' Number of training samples out of range 0-20')
       return
      endif
      if ((unc .lt. 0) .or. (unc .gt. 20)) then
       goflag = .false.
       write(*,40)
40     format(' Number of verification samples out of range 0-20')
       return
      endif
c
c repoint the spectrum description file pointer to the
c beginning of the spectrum list
c
      rewind 1
      read(1,30) dummy
      read(1,30) dummy
c
c both files are kept open. another subroutine will use them
c to load various arrays
c
      return
      end


c
c Subroutine Lstat computes the mean and variance of each column
c the loading matrix, L.
c
        subroutine Lstat(nc,nf,L,MeanL,VarL)
        integer nc,nf
```

```
        real*8 L(nc,nc),MeanL(nc),VarL(nc)
        real*8 sumL,sumLsq
        do 99998 j=1,nf
          sumL=0
          sumLsq=0
          do 99999 i=1,nc
            sumL=sumL+L(i,j)
            sumLsq=sumLsq+L(i,j)**2
99999     continue
          MeanL(j)=sumL/nc
          VarL(j)=(sumLsq-sumL**2/nc)/(nc-1)
99998   continue
        return
        end
c
c Subroutine MLR2 computes the coefficients of the MLR model
c for each property of the chemical training mixtures.  These
c are put into the matrix B2prop which has nc+1 rows but only nf+1
c are used for initial model estimates.
c
        subroutine MLR2(nprop,nc,ncp1,nf,pname,prop,L2
     1  ,B2prop,B2punc,pest,sprop)
        integer nprop,nc,ncp1,nf,nfp1,ndim
        real*4 prop(nc,nprop),pest(nc,nprop)
        real*8 L2(nc,ncp1),B2prop(ncp1,nprop),B2punc(ncp1,nprop)
        real*8 sprop(nprop),L2TL2(21,21),L2TL2I(21,21)
        real*8 ident(21,21)
        character*30 pname(nprop)
        write(*,1)
1       format(' Finding coefficients of mult. linear regr....'/)
        nfp1=nf+1
c write(*,4)
c4      format(' Matrix L2 transpose L2 (1st nf+1 elements only):')
c
c Form L2 transpose times L2 (L2TL2).  Only form the first nf+1 rows
c and columns since they are all that will be used in the model.
c Also temporarily assign the same elements to the inverse of
c L2 transpose L2 (L2TL2I).
c
        do 99997 i=1,nfp1
          do 99998 j=1,nfp1
            L2TL2(i,j)=0
            do 99999 k=1,nc
              L2TL2(i,j)=L2TL2(i,j)+L2(k,i)*L2(k,j)
99999       continue
            L2TL2I(i,j)=L2TL2(i,j)
99998     continue
c   write(*,5) (L2TL2(i,k),k=1,nfp1)
c5        format(50(1pe11.3))
99997   continue
c pause 'Hit return to continue'
c
c Invert L2TL2I to form the inverse of L2TL2.
c
        ndim=21
        call matinv(ndim,nfp1,L2TL2I)
c write(*,6)
c6      format(' Matrix L2 transpose L2 inverted:')
c do (i=1,nfp1)
c   write(*,7) (L2TL2I(i,j),j=1,nfp1)
c7        format(50(1pe11.3))
c continue
c write(*,10)
c10     format(' Product of L2 transpose L2 with its inverse:')
c do (i=1,nfp1)
```

```
c   do (j=1,nfp1)
c     ident(i,j)=0
c     do (k=1,nfp1)
c       ident(i,j)=ident(i,j)+L2TL2(i,k)*L2TL2I(k,j)
c     continue
c   continue
c   write(*,11) (ident(i,j),j=1,nfp1)
c11       format(50(0PF6.2))
c continue
c pause 'Hit return to continue'
        call formB2(nprop,nc,ncp1,nfp1,L2,L2TL2I,pname,prop
     1  ,B2prop,B2punc,pest,sprop)
        return
        end
c
c Subroutine MLR3 does multiple linear regression on the property
c passed to it with a reduced set of loading factors.  The reduction
c is based on the correlation of the property with the loading
c factors.  Those loadings with low correlation with the property
c are eliminated.  The reduced loading factor matrix is L3.
c
        subroutine MLR3(nprop,nc,ncp1,nf,pname,prop,L2,rhopL,rhothr
     1  ,B3prop,B3punc,pest,np,nloadp,L3,sprop)
        integer nprop,nc,ncp1,nf,np(nprop),nloadp(ncp1,nprop)
        integer col,ndim,inp
        real*4 prop(nc,nprop),pest(nc,nprop)
        real*8 L2(nc,ncp1),rhopL(nc,nprop),rhothr(nprop)
        real*8 B3prop(ncp1,nprop),sprop(nprop)
        real*8 B3punc(ncp1),L3(nc,ncp1,nprop)
        real*8 L3TL3(21,21),L3TL3I(21,21)
        real*8 ident(21,21)
        character*30 pname(nprop)
        write(*,1)
1       format(' Finding coefficients of mult. linear regr....'/)
        do 99997 iprop=1,nprop
          do 99998 i=1,nc
            do 99999 j=1,ncp1
              L3(i,j,iprop)=L2(i,j)
99999       continue
99998     continue
99997   continue
        call reduce(nprop,nc,ncp1,nf,L3,pname,rhopL,rhothr
     1  ,np,nloadp)
        do 99993 iprop=1,nprop
c   write(*,4)
c4        format(' Matrix L3 transpose L3 (1st np elements only):')
c
c   Form L3(row,col,iprop) transpose times L3(row,col,iprop) (L3TL3).
c   Only form the first np rows and columns since they are all that
c   will be used in the model.  Also temporarily assign the same
c   elements to the inverse of L3(row,col,iprop) transpose L3(row,col,iprop)
c   (L3TL3I).
c
          do 99994 i=1,np(iprop)
            do 99995 j=1,np(iprop)
              L3TL3(i,j)=0
              do 99996 k=1,nc
                L3TL3(i,j)=L3TL3(i,j)+L3(k,i,iprop)*L3(k,j,iprop)
99996         continue
              L3TL3I(i,j)=L3TL3(i,j)
99995       continue
c     write(*,5) (L3TL3(i,k),k=1,np(iprop))
c5          format(50(1pe11.3))
99994     continue
c   pause 'Hit return to continue'
```

```
c
c   Invert L3TL3I to form the inverse of L3TL3.
c
c   write(*,3)
c3        format(' Inverting L3 transpose L3...')
          ndim=21
          inp=np(iprop)
          call matinv(ndim,inp,L3TL3I)
c   write(*,6)
c6        format(' Matrix L3 transpose L3 inverted:')
c   do (i=1,np(iprop))
c     write(*,7) (L3TL3I(i,j),j=1,np(iprop))
c7          format(50(1pe11.3))
c   continue
c   write(*,10)
c10       format(' Product of L3 transpose L3 with its inverse:')
c   do (i=1,np(iprop))
c     do (j=1,np(iprop))
c       ident(i,j)=0
c       do (k=1,np(iprop))
c         ident(i,j)=ident(i,j)+L3TL3(i,k)*L3TL3I(k,j)
c       continue
c     continue
c     write(*,11) (ident(i,j),j=1,np(iprop))
c11         format(100(0PF6.2))
c   continue
c   pause 'Hit return to continue'
          col=iprop
          call form83(col,nprop,nc,ncp1,np,L3,L3TL3I,pname
     1    ,prop,B3prop,B3punc,pest,sprop)
          pause 'Hit return to continue'
99993   continue
        return
        end
c
c Subroutine propst computes the means and variances of the properties
c passed to it.
c
        subroutine propst(nprop,nc,prop,Meanp,Varp)
        integer nprop,nc
        real*4 prop(nc,nprop)
        real*8 sump,sumpsq,Meanp(nprop),Varp(nprop)
c
c Var(x) = E(x**2)-[E(x)]**2
c A computationally simpler form for Sample Variance of (x) is
c
c         s**2(x)=(Sum of x**2-(Sum of x)^2/n)/(n-1)
c
        do 99998 iprop=1,nprop
          sump=0
          sumpsq=0
          do 99999 i=1,nc
            sump=sump+prop(i,iprop)
            sumpsq=sumpsq+prop(i,iprop)**2
99999     continue
          Meanp(iprop)=sump/nc
          Varp(iprop)=(sumpsq-sump**2/nc)/(nc-1)
99998   continue
        return
        end
c
c Subroutine reduce chooses the loadings that will be used based
c on their correlation with the property, prop, passed to it.
c
        subroutine reduce(nprop,nc,ncp1,nf,L3,pname,rhopL,rhothr
```

```
     1   ,np,nloadp)
         integer nprop,nc,ncp1,nf,np(nprop),ncend
         integer nloadp(ncp1,nprop)
         real*8 L3(nc,ncp1,nprop),rhopL(nc,nprop),rhothr(nprop)
         character*30 pname(nprop)
         do 99999 iprop=1,nprop
           write(*,10) pname(iprop)
10         format(' For ',a,' the loading columns
     1 with high correlation are:')
           np(iprop)=1
           nloadp(np(iprop),iprop)=1
           j=2
           ncend=nf+1

           if (j .gt. ncend) goto 99998
c          do while (j.le.ncend)
99997      continue
             if (abs(rhopL(j-1,iprop)).ge.rhothr(iprop)) then
               np(iprop)=np(iprop)+1
               nloadp(np(iprop),iprop)=j
               write(*,20) nloadp(np(iprop),iprop)
20             format(' #',i3)
             else
               ncend=ncend-1
               if (j.le.ncend) then
                 do 99995 jj=j,ncend
                   do 99996 i=1,nc
                     L3(i,jj,iprop)=L3(i,jj+1,iprop)
99996              continue
99995            continue
               endif
             endif
             j=j+1
             if (j .le. ncend) goto 99997
99998      continue


99999    continue
         return
         end
c
c Subroutine repm reports to the user, on the screen, the matrix,
c 'mat', passed to it.  The numbers of rows and columns are passed in
c 'rows' and 'cols'.  The data in 'mat' must be typed real*4.
c
         subroutine repm(rows,cols,mat)
         integer rows,cols
         real*4 mat(rows,cols)
         do 99999 i=1,rows
           write(*,10) (mat(i,j),j=1,cols)
10         format(100(1pe11.3))
99999    continue
         pause 'Hit return to continue'
         return
         end
c
c Subroutine repmd reports to the user, on the screen, the matrix,
c 'mat', passed to it.  The numbers of rows and columns are passed in
c 'rows' and 'cols'.  The data in 'mat' must be typed real*8.
c
         subroutine repmd(rows,cols,mat)
         integer rows,cols
         real*8 mat(rows,cols)
         do 99999 i=1,rows
           write(*,10) (mat(i,j),j=1,cols)
```

```
10        format(100(1pe11.3))
99999   continue
        pause 'Hit return to continue'
        return
        end
c
c Subroutine reptD reports on the screen the results of reading the
c concentration and spectral data.  It is used for diagnosis
c and would not ordinarily be used in the final version.
c
        subroutine reptD(nprop,nr,nc,pname,prop,D)
        integer nprop,nr,nc,rows
        real*4 prop(nc,nprop)
        real*8 D(nr,nc)
        character*30 pname(nprop)
        write(*,10)
10      format(' Properties:')
        do 99998 iprop=1,nprop
          write(*,20) pname(iprop)
20        format(1x,a)
          do 99999 inc=1,nc
            write(*,30) prop(inc,iprop)
30          format(1pe15.7)
99999     continue
          pause 'Hit return to continue'
99998   continue
        write(*,40)
40      format(' D matrix (up to 1st 40 rows):')
        rows=nr
        if (nr.gt.40) rows=40
        call repmd(rows,nc,D)
        return
        end
c
c Subroutine reptst reports the results of the empirical tests
c to the user on the screen.
c
        subroutine reptst(nc,nf,re,ie,xe,IND)
        integer nc,nf,ncm1
        real*8 re(nc),ie(nc),xe(nc),IND(nc)
        ncm1=nc-1
        do 99999 i=1,ncm1
          write(*,10) i,re(i),ie(i),xe(i),IND(i)
10        format(' n =',i3,' re =',1pe15.7,' ie =',1pe15.7
     1    ,' xe =',1pe15.7,' IND =',1pe15.7)
99999   continue
        pause 'Hit return to continue'
        return
        end
c
c Subroutine repv reports to the user, on the screen, the vector,
c 'vec', passed to it.  The numbers of elements is passed in
c 'rows'.  The data in 'vec' must be typed real*4.
c
        subroutine repv(rows,vec)
        integer rows
        real*4 vec(rows)
        do 99999 i=1,rows
          write(*,10) vec(i)
10        format(1pe11.3)
99999   continue
        pause 'Hit return to continue'
        return
        end
c
```

```
c Subroutine repvd reports to the user, on the screen, the vector,
c 'vec', passed to it.  The numbers of elements is passed in
c 'rows'.  The data in 'vec' must be typed real*8.
c
        subroutine repvd(rows,vec)
        integer rows
        real*8 vec(rows)
        do 99999 i=1,rows
          write(*,10) vec(i)
10        format(1pe11.3)
99999   continue
        pause 'Hit return to continue'
        return
        end
c
c Subroutine tests computes the estimates of three types of error mentioned
c in Malinowski's paper; real error (re), imbedded error (ie) and extracted
c error (xe) plus an 'indicator function' IND which is related to the
c real error.  On the basis of these error estimates one can make a first
c conservative cut of the number of eigenvalues and eigenvectors used
c in the MLR.  The number of eigenvectors retained
c is nf.
c
        subroutine tests(nr,nc,lambda,re,ie,xe,IND,nf)
        integer nr,nc,nf
        real*8 lambda(nc),re(nc),ie(nc),xe(nc),IND(nc)
        write(*,1)
1       format(' Testing for number of eigenvalues to retain...'/)
        ncm1=nc-1
        do 99998 nf=1,ncm1
          sum=0
          nfp1=nf+1
          do 99999 i=nfp1,nc
            sum=sum+lambda(i)
99999     continue
          re(nf)=sqrt(sum/(nr*(nc-nf)))
          ie(nf)=sqrt(nf*sum/(nr*nc*(nc-nf)))
          xe(nf)=sqrt(sum/(nr*nc))
          IND(nf)=re(nf)/(nc-nf)**2
99998   continue
        nf=int(0.625*nc)
        write(*,20) nf
20      format(' No. of primary eigenvalues retained is ',i3)
        return
        end

c subroutine writemat
c
c writes selected data do calibration matrix file
c
c Input data (parameters):
c
c nprop   The number of "properties" for which to find models (<=4).
c         A property, in our case, is a chemical concentration.
c
c nc      The number of different concentration combinations (mixtures) used
c         in the training set (<=20).
c
c nr      The number of wavenumbers used from each spectrum of
c         Raman responses for training (<=25).
c
c lambda  A vector of the eigenvalues of Z with nc elements.
c
c F       A matrix whose columns are the abstract "factors" or
c         "eigenspectra" of D.  It is the same size as D.
```

```
c
c np      A vector of integers.  Each element is the number of columns of
c         the augmented loading matrix, L2, which are kept after examining
c         the correlation between as property and the loading matrix columns.
c         It has nprop elements.
c         columns but only the first np rows in each column are significant.
c
c nloadp  A matrix of integers.  Each column contains the subscripts
c         designating the loading columns of L2 which are retained
c         for a property to form L3.  Its size is (nc+1) rows by nprop
c         columns but only the first np (for a particular property)
c         elements in a particular column are significant.
c
c ncp1    columns in augmented loading matrix
c
c cell    cell number
c
c B3prop  A reduced matrix of the coefficients in the regression models used
c         to predict properties.  Each column contains the set of
c         coefficients for one property.  Its size is nc rows by nprop
c         columns but only the first np rows in each column are signficant.
c
c Output data:
c  file:
c  matrixy.dat - y corresponds to cell number. File contains the data
c                required by the control system to deduce the
c                composition of an unknown mixture, given its spectrum.
c
c
c Subprograms called: None
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine writemat(cell,nprop,nr,ncp1,nc,f,lambda,np,nloadp
     1,B3prop)
      integer nprop,nr,ncp1,nc,cell
      real*8 f(nr,nc),lambda(nc),B3prop(ncp1,nprop)
      integer np(nprop),nloadp(ncp1,nprop)
      character*13 matfile

      if (cell .eq. 0) matfile = 'matrix0.dat'
      if (cell .eq. 1) matfile = 'matrix1.dat'
      if (cell .eq. 2) matfile = 'matrix2.dat'
      if (cell .eq. 3) matfile = 'matrix3.dat'
      if (cell .eq. 4) matfile = 'matrix4.dat'
      open(1,file=matfile)
      close(1,status='delete')
      open(1,file=matfile,status='new')
      write(1,500) nc
500   format(1x,I3)
501   format(1x,E14.8)
      do 99999 i=1,nr
       do 99998 j=1,nc
        write(1,501) f(i,j)
99998  continue
99999 continue
      do 99997 i=1,nc
99997 write(1,501) lambda(i)
      do 99996 i=1,nprop
99996 write(1,500) np(i)
      do 99995 i=1,ncp1
       do 99994 j=1,nprop
        write(1,500) nloadp(i,j)
99994  continue
99995 continue
      do 99993 i=1,ncp1
```

```
       do 99992 j=1,nprop
        write(1,501) B3prop(i,j)
99992  continue
99993 continue
      close(1)

      return
      end



      subroutine rs(nm,n,a,w,matz,z,fv1,fv2,ierr)
c
      integer n,nm,ierr,matz
      double precision a(nm,n),w(n),z(nm,n),fv1(n),fv2(n)
c
c     this subroutine calls the recommended sequence of
c     subroutines from the eigensystem subroutine package (eispack)
c     to find the eigenvalues and eigenvectors (if desired)
c     of a real symmetric matrix.
c
c     on input
c
c        nm  must be set to the row dimension of the two-dimensional
c        array parameters as declared in the calling program
c        dimension statement.
c
c        n  is the order of the matrix  a.
c
c        a  contains the real symmetric matrix.
c
c        matz  is an integer variable set equal to zero if
c        only eigenvalues are desired.  otherwise it is set to
c        any non-zero integer for both eigenvalues and eigenvectors.
c
c     on output
c
c        w  contains the eigenvalues in ascending order.
c
c        z  contains the eigenvectors if matz is not zero.
c
c        ierr  is an integer output variable set equal to an error
c           completion code described in the documentation for tqlrat
c           and tql2.  the normal completion code is zero.
c
c        fv1  and  fv2  are temporary storage arrays.
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
c     ------------------------------------------------------------------
c
      if (n .le. nm) go to 10
      ierr = 10 * n
      go to 50
c
   10 if (matz .ne. 0) go to 20
c     ..... find eigenvalues only .....
      call  tred1(nm,n,a,w,fv1,fv2)
*  tqlrat encounters catastrophic underflow on the Vax
*     call  tqlrat(n,w,fv2,ierr)
      call  tql1(n,w,fv1,ierr)
```

```
      go to 50
c     ..... find both eigenvalues and eigenvectors .....
   20 call  tred2(nm,n,a,w,fv1,z)
      call  tql2(nm,n,w,fv1,z,ierr)
   50 return
      end
      subroutine tql1(n,d,e,ierr)
c
      integer i,j,l,m,n,ii,l1,l2,mml,ierr
      double precision d(n),e(n)
      double precision c,c2,c3,dl1,el1,f,g,h,p,r,s,s2,tst1,tst2,pythag
c
c     this subroutine is a translation of the algol procedure tql1,
c     num. math. 11, 293-306(1968) by bowdler, martin, reinsch, and
c     wilkinson.
c     handbook for auto. comp., vol.ii-linear algebra, 227-240(1971).
c
c     this subroutine finds the eigenvalues of a symmetric
c     tridiagonal matrix by the ql method.
c
c     on input
c
c        n is the order of the matrix.
c
c        d contains the diagonal elements of the input matrix.
c
c        e contains the subdiagonal elements of the input matrix
c          in its last n-1 positions.  e(1) is arbitrary.
c
c      on output
c
c        d contains the eigenvalues in ascending order.  if an
c          error exit is made, the eigenvalues are correct and
c          ordered for indices 1,2,...ierr-1, but may not be
c          the smallest eigenvalues.
c
c        e has been destroyed.
c
c        ierr is set to
c          zero       for normal return,
c          j          if the j-th eigenvalue has not been
c                     determined after 30 iterations.
c
c     calls pythag for  dsqrt(a*a + b*b) .
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
c     ------------------------------------------------------------------
c
      ierr = 0
      if (n .eq. 1) go to 1001
c
      do 100 i = 2, n
  100 e(i-1) = e(i)
c
      f = 0.0d0
      tst1 = 0.0d0
      e(n) = 0.0d0
c
      do 290 l = 1, n
         j = 0
         h = dabs(d(l)) + dabs(e(l))
```

```
         if (tst1 .lt. h) tst1 = h
c     ..... look for small sub-diagonal element .....
         do 110 m = l, n
            tst2 = tst1 + dabs(e(m))
            if (tst2 .eq. tst1) go to 120
c     ..... e(n) is always zero, so there is no exit
c                through the bottom of the loop .....
  110    continue
c
  120    if (m .eq. l) go to 210
  130    if (j .eq. 30) go to 1000
         j = j + 1
c     ..... form shift .....
         l1 = l + 1
         l2 = l1 + 1
         g = d(l)
         p = (d(l1) - g) / (2.0d0 * e(l))
         r = pythag(p,1.0d0)
         d(l) = e(l) / (p + dsign(r,p))
         d(l1) = e(l) * (p + dsign(r,p))
         dl1 = d(l1)
         h = g - d(l)
         if (l2 .gt. n) go to 145
c
         do 140 i = l2, n
  140    d(i) = d(i) - h
c
  145    f = f + h
c     ..... ql transformation .....
         p = d(m)
         c = 1.0d0
         c2 = c
         el1 = e(l1)
         s = 0.0d0
         mml = m - l
c     ..... for i=m-1 step -1 until l do -- .....
         do 200 ii = 1, mml
            c3 = c2
            c2 = c
            s2 = s
            i = m - ii
            g = c * e(i)
            h = c * p
            r = pythag(p,e(i))
            e(i+1) = s * r
            s = e(i) / r
            c = p / r
            p = c * d(i) - s * g
            d(i+1) = h + s * (c * g + s * d(i))
  200    continue
c
         p = -s * s2 * c3 * el1 * e(l) / dl1
         e(l) = s * p
         d(l) = c * p
         tst2 = tst1 + dabs(e(l))
         if (tst2 .gt. tst1) go to 130
  210    p = d(l) + f
c     ..... order eigenvalues .....
         if (l .eq. 1) go to 250
c     ..... for i=l step -1 until 2 do -- .....
         do 230 ii = 2, l
            i = l + 2 - ii
            if (p .ge. d(i-1)) go to 270
            d(i) = d(i-1)
  230    continue
```

```
c
  250    i = 1
  270    d(i) = p
  290 continue
c
      go to 1001
c     ..... set error -- no convergence to an
c                eigenvalue after 30 iterations .....
 1000 ierr = l
 1001 return
      end
      subroutine tql2(nm,n,d,e,z,ierr)
c
      integer i,j,k,l,m,n,ii,l1,l2,nm,mml,ierr
      double precision d(n),e(n),z(nm,n)
      double precision c,c2,c3,dl1,el1,f,g,h,p,r,s,s2,tst1,tst2,pythag
c
c     this subroutine is a translation of the algol procedure tql2,
c     num. math. 11, 293-306(1968) by bowdler, martin, reinsch, and
c     wilkinson.
c     handbook for auto. comp., vol.ii-linear algebra, 227-240(1971).
c
c     this subroutine finds the eigenvalues and eigenvectors
c     of a symmetric tridiagonal matrix by the ql method.
c     the eigenvectors of a full symmetric matrix can also
c     be found if  tred2  has been used to reduce this
c     full matrix to tridiagonal form.
c
c     on input
c
c        nm must be set to the row dimension of two-dimensional
c          array parameters as declared in the calling program
c          dimension statement.
c
c        n is the order of the matrix.
c
c        d contains the diagonal elements of the input matrix.
c
c        e contains the subdiagonal elements of the input matrix
c          in its last n-1 positions.  e(1) is arbitrary.
c
c        z contains the transformation matrix produced in the
c          reduction by  tred2, if performed.  if the eigenvectors
c          of the tridiagonal matrix are desired, z must contain
c          the identity matrix.
c
c      on output
c
c        d contains the eigenvalues in ascending order.  if an
c          error exit is made, the eigenvalues are correct but
c          unordered for indices 1,2,...,ierr-1.
c
c        e has been destroyed.
c
c        z contains orthonormal eigenvectors of the symmetric
c          tridiagonal (or full) matrix.  if an error exit is made,
c          z contains the eigenvectors associated with the stored
c          eigenvalues.
c
c        ierr is set to
c          zero       for normal return,
c          j          if the j-th eigenvalue has not been
c                     determined after 30 iterations.
c
c     calls pythag for  dsqrt(a*a + b*b) .
```

```
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
c     ------------------------------------------------------------------
c
      ierr = 0
      if (n .eq. 1) go to 1001
c
      do 100 i = 2, n
  100 e(i-1) = e(i)
c
      f = 0.0d0
      tst1 = 0.0d0
      e(n) = 0.0d0
c
      do 240 l = 1, n
         j = 0
         h = dabs(d(l)) + dabs(e(l))
         if (tst1 .lt. h) tst1 = h
c     ..... look for small sub-diagonal element .....
         do 110 m = l, n
            tst2 = tst1 + dabs(e(m))
            if (tst2 .eq. tst1) go to 120
c     ..... e(n) is always zero, so there is no exit
c                through the bottom of the loop .....
  110    continue
c
  120    if (m .eq. l) go to 220
  130    if (j .eq. 30) go to 1000
         j = j + 1
c     ..... form shift .....
         l1 = l + 1
         l2 = l1 + 1
         g = d(l)
         p = (d(l1) - g) / (2.0d0 * e(l))
         r = pythag(p,1.0d0)
         d(l) = e(l) / (p + dsign(r,p))
         d(l1) = e(l) * (p + dsign(r,p))
         dl1 = d(l1)
         h = g - d(l)
         if (l2 .gt. n) go to 145
c
         do 140 i = l2, n
  140    d(i) = d(i) - h
c
  145    f = f + h
c     ..... ql transformation .....
         p = d(m)
         c = 1.0d0
         c2 = c
         el1 = e(l1)
         s = 0.0d0
         mml = m - l
c     ..... for i=m-1 step -1 until l do -- .....
         do 200 ii = 1, mml
            c3 = c2
            c2 = c
            s2 = s
            i = m - ii
            g = c * e(i)
            h = c * p
            r = pythag(p,e(i))
```

```
            e(i+1) = s * r
            s = e(i) / r
            c = p / r
            p = c * d(i) - s * g
            d(i+1) = h + s * (c * g + s * d(i))
c     ..... form vector .....
            do 180 k = 1, n
               h = z(k,i+1)
               z(k,i+1) = s * z(k,i) + c * h
               z(k,i) = c * z(k,i) - s * h
  180       continue
c
  200    continue
c
         p = -s * s2 * c3 * el1 * e(l) / dl1
         e(l) = s * p
         d(l) = c * p
         tst2 = tst1 + dabs(e(l))
         if (tst2 .gt. tst1) go to 130
  220    d(l) = d(l) + f
  240 continue
c     ..... order eigenvalues and eigenvectors .....
      do 300 ii = 2, n
         i = ii - 1
         k = i
         p = d(i)
c
         do 260 j = ii, n
            if (d(j) .ge. p) go to 260
            k = j
            p = d(j)
  260    continue
c
         if (k .eq. i) go to 300
         d(k) = d(i)
         d(i) = p
c
         do 280 j = 1, n
            p = z(j,i)
            z(j,i) = z(j,k)
            z(j,k) = p
  280    continue
c
  300 continue
c
      go to 1001
c     ..... set error -- no convergence to an
c               eigenvalue after 30 iterations .....
 1000 ierr = l
 1001 return
      end
      subroutine tred1(nm,n,a,d,e,e2)
c
      integer i,j,k,l,n,ii,nm,jp1
      double precision a(nm,n),d(n),e(n),e2(n)
      double precision f,g,h,scale
c
c     this subroutine is a translation of the algol procedure tred1,
c     num. math. 11, 181-195(1968) by martin, reinsch, and wilkinson.
c     handbook for auto. comp., vol.ii-linear algebra, 212-226(1971).
c
c     this subroutine reduces a real symmetric matrix
c     to a symmetric tridiagonal matrix using
c     orthogonal similarity transformations.
c
```

```
c     on input
c
c        nm must be set to the row dimension of two-dimensional
c          array parameters as declared in the calling program
c          dimension statement.
c
c        n is the order of the matrix.
c
c        a contains the real symmetric input matrix.  only the
c          lower triangle of the matrix need be supplied.
c
c     on output
c
c        a contains information about the orthogonal trans-
c          formations used in the reduction in its strict lower
c          triangle.  the full upper triangle of a is unaltered.
c
c        d contains the diagonal elements of the tridiagonal matrix.
c
c        e contains the subdiagonal elements of the tridiagonal
c          matrix in its last n-1 positions.  e(1) is set to zero.
c
c        e2 contains the squares of the corresponding elements of e.
c          e2 may coincide with e if the squares are not needed.
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
c     ------------------------------------------------------------------
c
      do 100 i = 1, n
         d(i) = a(n,i)
         a(n,i) = a(i,i)
  100 continue
c     ..... for i=n step -1 until 1 do -- .....
      do 300 ii = 1, n
         i = n + 1 - ii
         l = i - 1
         h = 0.0d0
         scale = 0.0d0
         if (l .lt. 1) go to 130
c     ..... scale row (algol tol then not needed) .....
         do 120 k = 1, l
  120    scale = scale + dabs(d(k))
c
         if (scale .ne. 0.0d0) go to 140
c
         do 125 j = 1, l
            d(j) = a(l,j)
            a(l,j) = a(i,j)
            a(i,j) = 0.0d0
  125    continue
c
  130    e(i) = 0.0d0
         e2(i) = 0.0d0
         go to 300
c
  140    do 150 k = 1, l
            d(k) = d(k) / scale
            h = h + d(k) * d(k)
  150    continue
c
         e2(i) = scale * scale * h
```

```
         f = d(l)
         g = -dsign(dsqrt(h),f)
         e(i) = scale * g
         h = h - f * g
         d(l) = f - g
         if (l .eq. 1) go to 285
c     ..... form a*u .....
         do 170 j = 1, l
  170    e(j) = 0.0d0
c
         do 240 j = 1, l
            f = d(j)
            g = e(j) + a(j,j) * f
            jp1 = j + 1
            if (l .lt. jp1) go to 220
c
            do 200 k = jp1, l
               g = g + a(k,j) * d(k)
               e(k) = e(k) + a(k,j) * f
  200       continue
c
  220       e(j) = g
  240    continue
c     ..... form p .....
         f = 0.0d0
c
         do 245 j = 1, l
            e(j) = e(j) / h
            f = f + e(j) * d(j)
  245    continue
c
         h = f / (h + h)
c     ..... form q .....
         do 250 j = 1, l
  250    e(j) = e(j) - h * d(j)
c     ..... form reduced a .....
         do 280 j = 1, l
            f = d(j)
            g = e(j)
c
            do 260 k = j, l
  260       a(k,j) = a(k,j) - f * e(k) - g * d(k)
c
  280    continue
c
  285    do 290 j = 1, l
            f = d(j)
            d(j) = a(l,j)
            a(l,j) = a(i,j)
            a(i,j) = f * scale
  290    continue
c
  300 continue
c
      return
      end
      subroutine tred2(nm,n,a,d,e,z)
c
      integer i,j,k,l,n,ii,nm,jp1
      double precision a(nm,n),d(n),e(n),z(nm,n)
      double precision f,g,h,hh,scale
c
c     this subroutine is a translation of the algol procedure tred2,
c     num. math. 11, 181-195(1968) by martin, reinsch, and wilkinson.
c     handbook for auto. comp., vol.ii-linear algebra, 212-226(1971).
```

```
c
c     this subroutine reduces a real symmetric matrix to a
c     symmetric tridiagonal matrix using and accumulating
c     orthogonal similarity transformations.
c
c     on input
c
c        nm must be set to the row dimension of two-dimensional
c          array parameters as declared in the calling program
c          dimension statement.
c
c        n is the order of the matrix.
c
c        a contains the real symmetric input matrix.  only the
c          lower triangle of the matrix need be supplied.
c
c     on output
c
c        d contains the diagonal elements of the tridiagonal matrix.
c
c        e contains the subdiagonal elements of the tridiagonal
c          matrix in its last n-1 positions.  e(1) is set to zero.
c
c        z contains the orthogonal transformation matrix
c          produced in the reduction.
c
c        a and z may coincide.  if distinct, a is unaltered.
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
c     ------------------------------------------------------------------
c
      do 100 i = 1, n
c
         do 80 j = i, n
   80    z(j,i) = a(j,i)
c
         d(i) = a(n,i)
  100 continue
c
      if (n .eq. 1) go to 510
c     ..... for i=n step -1 until 2 do -- .....
      do 300 ii = 2, n
         i = n + 2 - ii
         l = i - 1
         h = 0.0d0
         scale = 0.0d0
         if (l .lt. 2) go to 130
c     ..... scale row (algol tol then not needed) .....
         do 120 k = 1, l
  120    scale = scale + dabs(d(k))
c
         if (scale .ne. 0.0d0) go to 140
  130    e(i) = d(l)
c
         do 135 j = 1, l
            d(j) = z(l,j)
            z(i,j) = 0.0d0
            z(j,i) = 0.0d0
  135    continue
c
         go to 290
```

```
c
  140    do 150 k = 1, l
            d(k) = d(k) / scale
            h = h + d(k) * d(k)
  150    continue
c
         f = d(l)
         g = -dsign(dsqrt(h),f)
         e(i) = scale * g
         h = h - f * g
         d(l) = f - g
c     ..... form a*u .....
         do 170 j = 1, l
  170    e(j) = 0.0d0
c
         do 240 j = 1, l
            f = d(j)
            z(j,i) = f
            g = e(j) + z(j,j) * f
            jp1 = j + 1
            if (l .lt. jp1) go to 220
c
            do 200 k = jp1, l
               g = g + z(k,j) * d(k)
               e(k) = e(k) + z(k,j) * f
  200       continue
c
  220       e(j) = g
  240    continue
c     ..... form p .....
         f = 0.0d0
c
         do 245 j = 1, l
            e(j) = e(j) / h
            f = f + e(j) * d(j)
  245    continue
c
         hh = f / (h + h)
c     ..... form q .....
         do 250 j = 1, l
  250    e(j) = e(j) - hh * d(j)
c     ..... form reduced a .....
         do 280 j = 1, l
            f = d(j)
            g = e(j)
c
            do 260 k = j, l
  260       z(k,j) = z(k,j) - f * e(k) - g * d(k)
c
            d(j) = z(l,j)
            z(i,j) = 0.0d0
  280    continue
c
  290    d(i) = h
  300 continue
c     ..... accumulation of transformation matrices .....
      do 500 i = 2, n
         l = i - 1
         z(n,l) = z(l,l)
         z(l,l) = 1.0d0
         h = d(i)
         if (h .eq. 0.0d0) go to 380
c
         do 330 k = 1, l
  330    d(k) = z(k,i) / h
```

```
c
         do 360 j = 1, l
            g = 0.0d0
c
            do 340 k = 1, l
  340       g = g + z(k,i) * z(k,j)
c
            do 360 k = 1, l
               z(k,j) = z(k,j) - g * d(k)
  360     continue
c
  380     do 400 k = 1, l
  400     z(k,i) = 0.0d0
c
  500 continue
c
  510 do 520 i = 1, n
         d(i) = z(n,i)
         z(n,i) = 0.0d0
  520 continue
c
      z(n,n) = 1.0d0
      e(1) = 0.0d0
      return
      end
      double precision function pythag(a,b)
      double precision a,b
c
c     finds dsqrt(a**2+b**2) without overflow or destructive underflow
c
      double precision p,r,s,t,u
      p = dmax1(dabs(a),dabs(b))
      if (p .eq. 0.0d0) go to 20
      r = (dmin1(dabs(a),dabs(b))/p)**2
   10 continue
         t = 4.0d0 + r
         if (t .eq. 4.0d0) go to 20
         s = r/t
         u = 1.0d0 + 2.0d0*s
         p = u*p
         r = (s/u)**2 * r
      go to 10
   20 pythag = p
      return
      end

       subroutine matinv(ndim,norder,mat)
c
c Lengths of ik and jk must be set big enough
c to accomodate the largest number of training
c samples anticipated.
c
      integer ndim,norder,i,j,k,ik(21),jk(21),L
      logical jloop,iloop
      double precision mat(ndim,ndim),mmag
      det=1.0d0
      do 89989 k=1,norder
c
c Find largest element mat(i,j) in rest of matrix.
c
          amax = 0.0d0
          jloop=.true.
89996       continue
            iloop=.true.
89997        continue
```

```
            do 89998 i=k,norder
              do 89999 j=k,norder
                amag=abs(amax)
              mmag=abs(mat(i,j))
                if ((amag-mmag).le.0.0d0) then
                amax=mat(i,j)
                ik(k)=i
                jk(k)=j
              endif
89999        continue
89998       continue
c
c Interchange rows and columns to put amax in mat(k,k).
c
            if (amax.eq.0.0d0) then
              det=0
              return
            endif
            i=ik(k)
            if (i.ge.k) iloop=.false.
            if (iloop) goto 89997
          if (i.ne.k) then
            do 89995 j=1,norder
                savenum=mat(k,j)
              mat(k,j)=mat(i,j)
              mat(i,j)=-savenum
89995       continue
            endif
          j=jk(k)
          if (j.ge.k) jloop=.false.
          if (jloop) goto 89996
        if (j.ne.k) then
          do 89994 i=1,norder
              savenum = mat(i,k)
            mat(i,k) = mat(i,j)
            mat(i,j) = -savenum
89994     continue
        endif
c
c Accumulate elements of inverse matrix.
c
      do 89993 i=1,norder
          if (i.ne.k) mat(i,k)=-mat(i,k)/amax
89993 continue
        do 89991 i=1,norder
          do 89992 j=1,norder
            if ((i.ne.k).and.(j.ne.k)) then
              mat(i,j)=mat(i,j)+mat(i,k)*mat(k,j)
            endif
89992     continue
89991   continue
        do 89990 j=1,norder
          if (j.ne.k) mat(k,j)=mat(k,j)/amax
89990   continue
          mat(k,k)=1/amax
        det=det*amax
89989 continue
c
c Restore ordering of matrix.
c
      do 89986 L=1,norder
          k=norder-L+1
        j=ik(k)
        if (j.gt.k) then
          do 89988 i=1,norder
```

```
            savenum=mat(i,k)
           mat(i,k)=-mat(i,j)
           mat(i,j)=savenum
89988     continue
         endif
           i = jk(k)
         if (i.gt.k) then
           do 89987 j=1,norder
              savenum=mat(k,j)
           mat(k,j)=-mat(i,j)
           mat(i,j)=savenum
89987     continue
         endif
89986 continue
      return
      end
```

```
/* UT Measurement and Control Engineering Center                          */
/*                                                                        */
/* Stephen W. Kercel                                        June 1, 1990 */
/*                                                                        */
/* specr0.c                                                               */
/*                                                                        */
/* Program polls the hardware and files. Then it moves the positioners to */
/* collect spectra from the selected cell.If not ready, it returns to DOS. */
/* If files and hardware are ready, program collects the spectra described */
/* described in spec0.dat.                                                */
/*                                                                        */
/* CAUTION:                                                               */
/* This program is not portable. It was developed under Microsoft Quick C, */
/* using Microsoft and National Instrument library functions.             */
/* The I/O messages are hardware dependent,                               */
/* being specifically designed for two Daedal PC23 indexers, a            */
/* Perkin-Elmer 1700 interferometer connected via a National Instruments  */
/* IEEE-488 board, and a TTL controlled solenoid run via the printer port. */
/*                                                                        */
/* DO NOT COMPILE THIS PROGRAM WITH ANY COMPILER EXCEPT MICROSOFT QUICK C */
/* VERSION 2.01 OR HIGHER.                                                */
/*                                                                        */
/* RUN THE EXECUTABLE CODE ONLY ON A PC EQUIPPED AS DESCRIBED ABOVE.      */
/* MAKE SURE THE I/O ADDRESS SWITCH SETTINGS ON THE BOARDS AGREE WITH     */
/* THE I/O ADDRESSES IN THE DECLARED CONSTANTS OF THIS PROGRAM.           */
/*                                                                        */
/* FAILURE TO OBSERVE THE ABOVE PRECAUTIONS WILL LEAD TO UNEXPECTED RESULTS*/
/*                                                                        */
/* Input Data:                                                            */
/*  Files created by user with text editor:                               */
/*   cella0.dat through cella4.dat - contain data for each cell           */
/*   spec0.dat - contains description of spectra to be collected          */
/*  Vector obtained from spectrometer:                                    */
/*   spectrum0 - spectrum of current sample                               */
/*                                                                        */
/* Output Data (sent to spectrum files):                                  */
/*  spectrum0 - one spectrum vector sent to each file                     */
/*                                                                        */
/* Functions called from main():                                          */
/*  pos_there() - Assures that PC23 boards are ready to accept commands   */
/*  set_mode() - sets PC23s to absolute positioning mode                  */
/*  dofiles() - initializes arrays from cell and matrix files             */
/*  gig1700() - assures that the spectrometer is there                    */
/*  flip_dump() - moves beam dump solenoid                                */
/*  move_stage() - move stage to next set of absolute coordinates         */
/*  get_spectrum() - collect spectrum through spectrometer                */
/*  file_spec() - writes spectrum vector to spectrum file                 */
/*  ungig1700() - return spectrometer to conventional mode                */
/*                                                                        */
/**************************************************************************/

/*headers*/
#include <stdio.h>  /* I/O headers standard for most C programs*/
#include <string.h> /* headers for string handling -- Microsoft library*/
#include <conio.h> /* I/O headers -- Microsoft library*/
#include <stdlib.h> /* Microsoft extensions */

/*defined constants*/
#define TRUE 1
#define FALSE 0
#define ADDRESS0 0x300 /*positioner 0 address*/
#define ADDRESS1 0x310 /*positioner 1 address*/
#define ADDRESS2 0x378 /*beam dump address*/
#define TIMEOUT 1200 /*maximum spectrum scan time in seconds*/
#define MAXINT 32000
```

```
#define RETURN 0xD
#define MAXSPEC 25 /*maximum number of spectrum points used by a cell*/
#define MAXCOMPS 4 /*maximum number of components in a cell*/
#define MAXSTR 30 /*maximum component string length*/
#define MAXSFILES 30 /*maximum number of spectrum files per cell*/

/* In addition to the general purpose defined constants listed above, */
/* the control of the positioner makes use of various status and control*/
/* bit masks. Pages 22-30 of the PC23-03 manual describe the functions of*/
/* each registers bits (with the exception of the status registers bit 7*/
/* which indicates if the PC23 buffer is over half full). Page 30 gives*/
/* a step by step procedure of the handshaking process. Page 62 explains*/
/* the differences between the binary write (time distance) function and*/
/* the normal write function.                                          */

#define FAIL 0x20 /* 0010 0000 */
#define START_M 0x7F /* 0111 1111 */
#define INTCLR 0x20 /* 0010 0000 */
#define RESTART 0x17 /* 0001 0111 */
#define POSITIVE 0x8000 /* 1000 0000 0000 0000 */
#define ALDONE 0x7 /* 0111 */
#define STOP 0x64 /* 0110 0100 */
#define START 0x40 /* 0100 0000 */
#define CB 0x60 /* 0110 0000 */
#define CB1 0x61 /* 0110 0001 */
#define IDB_M 0x10 /* 0001 0000 */
#define IDB 0x70 /* 0111 0000 */
#define IDB1 0x71 /* 0111 0001 */
#define ODB 0x8 /* 1000 */
#define ACK 0xE0 /* 1110 0000 */
#define BFULL 0x80 /* 1000 0000 */ /*checks buffer status*/

/* All preceding compiler directives are overhead. The actual program*/
/* starts here. No global variables have been declared.              */

main()
{
        /* declarations and initializations */

        int i,k;/*loop counters*/
        int l;/*axis pointer*/
        int j;/*array pointer -- cell number*/
        int modeflag = TRUE;/*1 => both positioners went home */
        int goflag = TRUE;/*1 => both positioners initialized */
        int quitflag = FALSE;/* TRUE => halt, return to DOS */
        int retflag;/*TRUE => program ran ok*/
        char string_array[5][MAXCOMPS][MAXSTR];/*composition strings*/
        int cellflag[5];/*TRUE => cell is active*/
        int scans[5];/*number of scans used for current cell*/
        int scount[5];/*number of points in spectrum of current cell*/
        float norm[5];/*normalization indicator for current cell*/
        int spect[5][MAXSPEC];/*spectrum wave numbers for current cell*/
        int a = TRUE;/*TRUE => all files opened and had good data*/
        int g[5] = {0,0,0,0,0};/*number of components in this cell's mixture*/
        long cellarray0[5][3];/*excite positioner coordinate vectors*/
        long cellarray1[5][3];/*receive positioner coordinate vectors*/
        float spectrum0[MAXSPEC];/*spectrum vector*/
        float normfact = 1.0;/*height drift normalizing factor*/
        int z;/*TRUE => spectrum collected ok*/
        int specfs;/*number of spectra to be obtained*/
        char s_arr[MAXSFILES][13];/*spectrum file names*/
        int tvf[MAXSFILES];/*holds training verification flags*/
        float f_arr[MAXSFILES][MAXCOMPS];/*composition vectors*/
        int specok = TRUE;/*TRUE => spectrum ok*/
        char keybuf[81];/*keyboard input buffer*/
```

```
        /* the function starts doing work here */

        pos_there(&goflag); /* check whether positioners are there */
        if (goflag)
         {
          set_mode(&modeflag); /*send positioners home*/
          goflag = goflag && modeflag;
         }
        if (goflag)
         {
          a = do_files(cellflag,cellarray0,cellarray1,g,string_array,
            scans,scount,norm,spect,&j,&specfs,s_arr,tvf,f_arr);
            /*get data from files*/
          goflag = goflag && a;
         }
        if (goflag)
         {
          a = gig1700(); /*gig the spectrometer*/
          goflag = goflag && a;
         }

        if (goflag) /*if everything is ready*/
         {
          flip_dump(1); /*move beam dump to block laser*/
          move_stage(j,ADDRESS0,cellarray0);/*excitation stage*/
          flip_dump(0); /*move beam dump to unblock laser*/
          move_stage(j,ADDRESS1,cellarray1);/*receive stage*/
          for (i=0;((i<specfs) && (a));i++)
           {
            specok = FALSE;
            while (!specok)
             {
              z = get_spectrum(j,scount[j],spect,scans[j],norm[j],
                                normfact,spectrum0);
              printf("\n<S>ave spectrum or <R>un it again? ");
              scanf("%s",keybuf);
              fflush(stdin);
              if ((*keybuf == 'S') || (*keybuf == 's'))
               specok = TRUE;
             }
            a = file_spec(i,s_arr,scount[j],spectrum0);
            a = (a && z);
           }
          a = ungig1700();/*return spectrometer to conventional mode*/
          retflag = TRUE;
         }
        else
         retflag = FALSE;
        return(retflag);
}


/* First Level - Functions First Called by  Main Functions              */

/* function flip_dump()                                                 */
/*                                                                      */
/* Function to operate the beam dump solenoid.                          */
/* The beam dump solenoid is controlled by the logic state on pin 2 of the*/
/* parallel printer port. A byte is written to the port address 378 hex.  */
/* States of pins 2 through 9 on the printer port connector correspond to */
/* the states of bits 0 through 7 resectively of the output byte. Sending */
/* a 0 to the port latches pin 2 low. Sending a 1 to the port latches pin */
/* 2 high.                                                              */
/*                                                                      */
```

```
/* CAUTION: DO NOT USE THIS FUNCTION IF THERE IS A PRINTER CONNECTED TO   */
/* THE PARALLEL PRINTER ADAPTER.                                          */
/*                                                                        */
/* input data (parameter):                                                */
/*  flag - 1 => energize pin 2                                            */
/*                                                                        */
/* output data (sent to printer port):                                    */
/*  flag - 1 => energize pin 2                                            */
/*                                                                        */
/* functions called: none                                                 */
/**************************************************************************/

flip_dump(flag)
int flag;/*1 => energize pin 2*/
{
        outp(ADDRESS2,flag);
}

/* function get_spectrum()                                                */
/*                                                                        */
/* function collects a spectrum from the interferometer                   */
/*                                                                        */
/* function sends the number of scans to the interferometer, starts the   */
/* scanning operation, polls the interferometer until either the scanning */
/* is completed or the TIMEOUT constant (seconds) is exceeded, downloads  */
/* the collected spectrum to memory, selects the desired spectrum points, */
/* drift normalizes the spectrum points if necessary, and returns the     */
/* short spectrum vector to the calling function.                         */
/*                                                                        */
/* NOTE: ACTUAL IMPLEMENTATION OF INTERFEROMETER FUNCTIONS AWAITS FURTHER */
/* TECHNICAL INFORMATION FROM PERKIN-ELMER. FOR THE MEAN TIME, DUMMY      */
/* VERSIONS OF THE FUNCTIONS HAVE BEEN IMPLEMENTED.                       */
/*                                                                        */
/* input data:                                                            */
/*  parameters:                                                           */
/*   cell - cell number                                                   */
/*   cols - number of spectrum points to be returned                      */
/*   spect - list of wave numbers for which intensities are sought        */
/*   scans - number of scans to be used in collection of spectrum         */
/*   normflag - >0.1 => normalize for height drift                        */
/*   normfact - multiply each point by this factor if drift normalizing   */
/*  from interferometer via GPIB bus:                                     */
/*   long_spec - long version of spectrum                                 */
/*                                                                        */
/* output data:                                                           */
/*  parameter: spectrum0 - short version of normalized spectrum vector    */
/*  call return: z - TRUE => spectrum collected OK                        */
/*                                                                        */
/* functions called: none yet                                             */
/*                                                                        */
/**************************************************************************/

get_spectrum(cell,cols,spect,scans,normflag,normfact,spectrum0)
int cell;/*cell number*/
int cols;/*number of components in spectrum*/
int spect[][MAXSPEC];/*wave numbers for which intensities are sought*/
int scans;/*number of scans in spectrum*/
float normflag;/* >0.1 => normalize for height drift*/
float normfact;/*normalizing factor*/
float spectrum0[];/*spectrum vector*/

{
        int i;
        for (i=0;i<cols;i++)
         if (i == 2)
```

```
                spectrum0[i] = 14.0;
        else
                spectrum0[i] = 0.1 * i;
        return(TRUE);

}


/* function pos_there                                                    */
/*                                                                       */
/* initializes both positioners and makes sure they are both there       */
/*                                                                       */
/* input data:                                                           */
/*  readflag - flag passed back by initialize()                          */
/*                                                                       */
/* output data:                                                          */
/*  gofrag - flag passsed back to calling function                       */
/*                                                                       */
/* functions called:                                                     */
/*  initialize() - initializes addressed positioner                      */
/*                                                                       */
/***********************************************************************/

pos_there(gofrag)
int *gofrag;/*TRUE => Both positioners initialized*/

{
        int readflag = TRUE;/* TRUE => positioner initialized */
        int address;/*address of positioner*/

        address = ADDRESS0;
        initialize(address,&readflag);/*set up PC23*/
        *gofrag = readflag;
        if (!readflag)
         printf("\nPositioner %x failed to initialize\n",address);
        address = ADDRESS1;
        initialize(address,&readflag);
        *gofrag = *gofrag && readflag;
        if (!readflag)
         printf("\nPositioner %x failed to initialize\n",address);


}


/* function set_mode                                                     */
/*                                                                       */
/* homes both positioners and sets to absolute positioning mode          */
/*                                                                       */
/* input data:                                                           */
/*  readflag - flag passed back by set_one()                             */
/*                                                                       */
/* output data:                                                          */
/*  gofrag - flag passsed back to calling function                       */
/*                                                                       */
/* functions called:                                                     */
/*  set_one() - send messages to addressed positioner                    */
/*                                                                       */
/***********************************************************************/

set_mode(gofrag)
int *gofrag;/*TRUE => Both positioners went home*/

{
        int readflag = TRUE;/* TRUE => positioner went home */
```

```
        int address;/*address of positioner*/

        address = ADDRESS0;
        set_one(address,&readflag);/*set up PC23*/
        *gofrag = readflag;
        if (!readflag)
         printf("\nPositioner %x failed to go home\n",address);
        address = ADDRESS1;
        set_one(address,&readflag);
        *gofrag = *gofrag && readflag;
        if (!readflag)
         printf("\nPositioner %x failed to go home\n",address);

}

/* function gig1700()                                                    */
/*                                                                       */
/* function to set up the interferometer                                 */
/*                                                                       */
/* function causes the control computer to assume control of the IEEE-488 */
/* bus, set up the interferometer, set it to ASCII mode, and query for an */
/* acknowledgment from the interferometer.                               */
/*                                                                       */
/* NOTE: ACTUAL IMPLEMENTATION OF INTERFEROMETER FUNCTIONS AWAITS FURTHER */
/* TECHNICAL INFORMATION FROM PERKIN-ELMER. FOR THE MEAN TIME, DUMMY     */
/* VERSIONS OF THE FUNCTIONS HAVE BEEN IMPLEMENTED.                      */
/*                                                                       */
/* Input data: none                                                      */
/*                                                                       */
/* Output data:                                                          */
/*  z - TRUE => interferometer responded as expected                     */
/*                                                                       */
/* Functions called: none                                                */
/*                                                                       */
/*************************************************************************/

gig1700()
{
        return(TRUE);
}

/* function ungig1700()                                                  */
/*                                                                       */
/* function to return interferometer to conventional mode                */
/*                                                                       */
/* function causes the control computer to relinquish control of the GPIB */
/* bus, set interferometer to binary mode, and return control to the     */
/* interferometer controller keyboard.                                   */
/*                                                                       */
/* NOTE: ACTUAL IMPLEMENTATION OF INTERFEROMETER FUNCTIONS AWAITS FURTHER */
/* TECHNICAL INFORMATION FROM PERKIN-ELMER. FOR THE MEAN TIME, DUMMY     */
/* VERSIONS OF THE FUNCTIONS HAVE BEEN IMPLEMENTED.                      */
/*                                                                       */
/* Input data: none                                                      */
/*                                                                       */
/* Output data:                                                          */
/*  z - TRUE => interferometer responded as expected                     */
/*                                                                       */
/* Functions called: none                                                */
/*                                                                       */
/*************************************************************************/

ungig1700()
{
        return(TRUE);
```

```
}

/* function move_stage()                                                    */
/*                                                                          */
/* Executes absolute translation of stage in 3 dimensions                   */
/*                                                                          */
/* Before this function is invoked, other functions have initialized the    */
/* positioner and send commands setting the home position, velocity,        */
/* and acceleration. The mode has been preset to absolute displacement      */
/* with respect to home coordinates.                                        */
/*                                                                          */
/* The function reveives a copy of the cell number, the motor controller    */
/* address, and a pointer to the to the array of coordinates, from the      */
/* calling function. It selects the three appropriate coordinates from the */
/* array, formulates the displacement strings, sends the strings to the     */
/* positioner, and sends the "go" message to the positioner.                */
/*                                                                          */
/* input data (parameters from calling function):                           */
/*  jj - cell number                                                        */
/*  addr - I/O address of motor controller                                  */
/*  coords - array of positioniing coordinates                              */
/*                                                                          */
/* output data (parameter to called function):                              */
/*  s - string sent to positioner via writecmd()                            */
/*  Note: actual output is new location of positioner                       */
/*                                                                          */
/* functions called:                                                        */
/*  writecmd() - sends string to positioner                                 */
/*                                                                          */
/**************************************************************************/

move_stage(jj,addr,coords)
int jj;/*cell number*/
int addr;/*I/O address of motor controller*/
long coords[][3];/*list of coordinates*/

{
        char s[81];/*string sent to controller*/
        int i;/*loop counter*/
        char s1[17];/*axis portion of string*/
        char s2[2] = "D";/*command portion of string*/
        char s3[17];/*actual displacement in motor steps*/
        char s4[81];/*positioner response buffer*/
        long temp;/*temporary coordinate placeholder*/

        for (i=1;i<=3;i++) /*i is axis number */
         {
          temp = coords[jj][i-1];
          ltoa(temp,s3,10);
          itoa(i,s1,10);
          strcpy(s,s1);
          strcat(s,s2);
          strcat(s,s3);
          strcat(s," ");
          printf("\nString %s sent to positioner\n",s);
          writecmd(addr,s);  /*send displacement to positioner*/
          }
        strcpy(s,"G123 ");
        printf("\nString %s sent to positioner\n",s);
        writecmd(addr,s); /*make the stage go*/
        writecmd(addr," 1PR ");
        readanswer(addr,s4);
        printf("\nAxis 1 is at %s\n",s4);
        writecmd(addr," 2PR ");
        readanswer(addr,s4);
```

```
        printf("\nAxis 2 is at %s\n",s4);
        writecmd(addr," 3PR ");
        readanswer(addr,s4);
        printf("\nAxis 3 is at %s\n",s4);
}

/* function do_files()                                                    */
/*                                                                        */
/* function initializes data arrays from external files                   */
/*                                                                        */
/* Input data (parameter arrays received from lower level functions):     */
/*  f = cell status vector -- TRUE => cell is active                      */
/*  a0 = excitation positioner coordinate vectors                         */
/*  a1 = receiving positioner coordinate vectors                          */
/*  g = number of components in this cell's mixture                       */
/*  string_array = composition strings                                    */
/*  scans = number of scans used for current cell                         */
/*  scount = number of points in spectrum of current cell                 */
/*  norm = normalization indicator for current cell                       */
/*  spect = spectrum wave numbers for current cell                        */
/*  cell0 = cell in which spectra are taken                               */
/*  specfs0 = number of spectra to be taken                               */
/*  s_arr = filenames containing spectra                                  */
/*  tvf = training verification flags                                     */
/*  f_arr = list of composition vectors                                   */
/*                                                                        */
/* Output data (parameter arrays sent to calling function):               */
/*  f = cell status vector -- TRUE => cell is active                      */
/*  a0 = excitation positioner coordinate vectors                         */
/*  a1 = receiving positioner coordinate vectors                          */
/*  g = number of components in this cell's mixture                       */
/*  string_array = composition strings                                    */
/*  scans = number of scans used for current cell                         */
/*  scount = number of points in spectrum of current cell                 */
/*  norm = normalization indicator for current cell                       */
/*  spect = spectrum wave numbers for current cell                        */
/*  cell0 = cell in which spectra are taken                               */
/*  specfs0 = number of spectra to be taken                               */
/*  s_arr = filenames containing spectra                                  */
/*  tvf = training verification flags                                     */
/*  f_arr = list of composition vectors                                   */
/*                                                                        */
/* Functions called:                                                      */
/*  cellfunc() - brings in data from cell files                           */
/*  feedback1() - prints cell file data to screen                         */
/*  specfunc() - reads data from spec).dat to memory                      */
/*  feedback3() - prints spectrum description to screen                   */
/*                                                                        */
/************************************************************************/

do_files(f,a0,a1,g,string_array,scans,scount,norm,spect,cell0,specfs0,
                s_arr,tvf,f_arr)
int f[];/*TRUE => cell is active*/
long a0[][3];/*excite positioner coordinate vectors*/
long a1[][3];/*receive positioner coordinate vectors*/
int g[];/*number of components in this cell's mixture*/
char string_array[][MAXCOMPS][MAXSTR];/*composition strings*/
int scans[];/*number of scans used for current cell*/
int scount[];/*number of points in spectrum of current cell*/
float norm[];/*normalization indicator for current cell*/
int spect[][MAXSPEC];/*spectrum wave numbers for current cell*/
int *cell0;/*cell in which spectra are taken*/
int *specfs0;/*number of spectra in this set*/
char s_arr[][13];/*list of filenames*/
int tvf[];/*training verification flags*/
```

```
float f_arr[][MAXCOMPS];/*composition vectors*/
{
        int a;/*TRUE => cell files ok*/
        int b;/*TRUE => description files ok*/
        int c;/*TRUE => data moved ok from files to arrays*/
        int cell;/*cell number input buffer*/
        int specfs;/*number of spectra input buffer*/

        a = cellfunc(f,a0,a1,g,string_array,scans,scount,norm,spect);
        feedback1(f,a0,a1,g,string_array,a,scans,scount,norm,spect);
        if (a)
         {
          b = specfunc(g,&cell,&specfs,s_arr,tvf,f_arr);
          if (b)
           {
            feedback3(g[cell],cell,specfs,s_arr,tvf,f_arr);
            *cell0 = cell;
            *specfs0 = specfs;
           }
         }
        if (!(f[cell]))
         {
          printf("\nSpecified cell is inactive\n");
          a = FALSE;
         }
        c = a && b;
        return(c);
}

/* function file_spec()                                                    */
/*                                                                         */
/* function reads data from spectrum array into file                       */
/*                                                                         */
/* function opens named file and writes spectral data then closes          */
/*                                                                         */
/* input data:                                                             */
/*  j - spectrum number to be filed                                        */
/*  count - number of points in spectrum                                   */
/*  string_arr - list of spectrum filenames                                */
/*  f_arr - spectrum vector                                                */
/*                                                                         */
/* output data (paramaters):                                               */
/*  list of spectrum values                                                */
/*                                                                         */
/* functions called: none                                                  */
/*                                                                         */
/*************************************************************************/

file_spec(j,string_arr,count,f_arr)
int j;/*spectrum number*/
int count;/*number of components in spectrum vector*/
char string_arr[][13];/*list of filenames*/
float f_arr[];/*spectrum vector*/
{
        /*local declarations and initializations*/
        int i;/*loop counters*/
        int d;/*conversion counter*/
        int c = TRUE;/*TRUE => all data has been gotten from file*/
        FILE *fj,*fopen();/*file handling pointers*/
        char sbuf[13];/*local string buffer*/

        /*function starts doing work here*/
        strcpy(sbuf,(&string_arr[0][0]) + (13*j));
        fj = fopen(sbuf,"w");
        if (fj != NULL) /*if file opened*/
```

```
        {
         for (i=0;((i < count) && (c));i++)
          {
           d = fprintf(fj,"%f\n",f_arr[i]);
           printf("%f\n",f_arr[i]);
           if (d == 0)
            {
             printf("\nElement %d did not go into %s\n",i,sbuf);
             c = FALSE;
            }
          }
         fclose(fj);
        }
       else
        {
         printf("\n%s failed to open\n",sbuf);
         c = FALSE;
        }
       return(c);
}

/* Second Level - Functions First Called by First Level Functions          */

/* function specfunc()                                                     */
/*                                                                         */
/* function reads data from speco.dat file into memory                     */
/*                                                                         */
/* function opens spec0.dat, reads the cell number and number of spectra,  */
/* then it reads the filename and composition vectors for the sets of      */
/* spectra of known mixtures                                               */
/*                                                                         */
/* input data:                                                             */
/*  spec0.dat - cell number, number of spectra, filenames, compositions    */
/*  g - vector of component counts per cell                                */
/*                                                                         */
/* output data (paramaters):                                               */
/*  cell - cell number in which spectra will be taken                      */
/*  specfs - number of spectra that will be taken                          */
/*  string_arr - list of spectrum filenames                                */
/*  tvf - training verification flag vector                                */
/*  f_arr - list of composition vectors                                    */
/*                                                                         */
/* functions called: none                                                  */
/*                                                                         */
/*************************************************************************/

specfunc(g,cell,specfs,string_arr,tvf,f_arr)
int g[];/*vector of number of components in mixture*/
int *cell;/*number of cell in which spectra will be taken*/
int *specfs;/*number of spectera to be obtained*/
char string_arr[][13];/*list of filenames*/
int tvf[];/*training verify flags*/
float f_arr[][MAXCOMPS];/*list of composition vectors*/
{
        /*local declarations and initializations*/
        int i,j;/*loop counters*/
        int b = 0;/*local count of spectra*/
        int a = 0;/*local cell number*/
        int d;/*conversion counter*/
        int c = TRUE;/*TRUE => all data has been gotten from file*/
        FILE *fj,*fopen();/*file handling pointers*/
        char sbuf[13];/*local string buffer*/
        float fbuf;/*local floating point variable buffer*/
        int components;/*component count*/
```

```
        /*function starts doing work here*/
        fj = fopen("spec0.dat","r");
        if (fj != NULL) /*if file opened*/
         {
          d = fscanf(fj,"%d%d",&a,&b);/*read first two entries*/
          if ((d == 2) && (a < 5) && (a > -1) && (b <= MAXSFILES))
           {
            components = g[a];
            *cell = a;
            *specfs = b;
            for (i=0;((i<b) && (c));i++)/*i points to the current spectrum*/
            /*as long as all goes well*/
            /*read next filename, flag, and composition vector from file*/
             {
              d = fscanf(fj,"%s",sbuf);
              if (d == 1)
               {
                strcpy((&string_arr[0][0]) + (13*i),sbuf);
                /*read the train/verify flag*/
                d = fscanf(fj,"%d",(&tvf[i]));
                if (d == 1)
                 {
                  for (j=0;((j<components) && (c));j++)
                  /*read each element of composition vector*/
                   {
                    d = fscanf(fj,"%f",&fbuf);
                    if (d == 1)
                     f_arr[i][j] = fbuf;
                    else
                     {
                      c = FALSE;
                      printf("\nComposition vector incorrectly read\n");
                     }
                   }
                 }
                else
                 {
                  c = FALSE;
                  printf("\nTraining verify flag incorrectly read\n");
                 }
               }
              else
               {
                c = FALSE;
                printf("\nSpectrum filename incorrectly read\n");
               }
             }
           }
          else
           {
            c = FALSE;
            printf("\nFirst two conversions in spec0.dat failed\n");
            printf("or cell number not between 0 and 4\n");
            printf("or number of spectra exceeds MAXSFILES\n");
           }
          fclose(fj);
         }
        else
         {
          printf("\nspec0.dat failed to open\n");
          c = FALSE;
         }
        return(c);
}
```

```
/* function feedback3()                                                    */
/*                                                                         */
/* function writes data obtained with specfunc to screen                   */
/*                                                                         */
/* input data(parameters):                                                 */
/*  components - number of components in mixture                           */
/*  cell - cell number                                                     */
/*  specfs - number of spectra                                             */
/*  string_arr - list of spectrum files                                    */
/*  tvf - training verification flag vector                                */
/*  f_arr - list of composition vectors                                    */
/*                                                                         */
/* output data(displayed to screen):                                       */
/*  components - number of components in mixture                           */
/*  cell - cell number                                                     */
/*  specfs - number of spectra                                             */
/*  string_arr - list of spectrum files                                    */
/*  tvf - training verification flag vector                                */
/*  f_arr - list of composition vectors                                    */
/*                                                                         */
/* functions called: none                                                  */
/*                                                                         */
/*************************************************************************/

feedback3(components,cell,specfs,string_arr,tvf,f_arr)
int components;/*number of components in mixture*/
int cell;/*number of cell in which spectra will be taken*/
int specfs;/*number of spectera to be obtained*/
char string_arr[][13];/*list of filenames*/
int tvf[];/*training verify flags*/
float f_arr[][MAXCOMPS];/*list of composition vectors*/
{
        /*local declarations and initializations*/
        int i,j;/*loop counters*/
        int b;/*local count of spectra*/
        int a;/*local cell number*/
        int d;/*conversion counter*/
        int c = TRUE;/*TRUE => all data has been gotten from file*/
        FILE *fj,*fopen();/*file handling pointers*/
        char sbuf[13];/*local string buffer*/
        float fbuf;/*local floating point variable buffer*/

        /*function starts doing work here*/
        printf("\nCell %d has %d components and %d spectra\n"
            ,cell,components,specfs);
        for (i=0;i < specfs; i++)
         {
          strcpy(sbuf,(&string_arr[0][0]) + (13*i));
          printf("\n%s\n",sbuf);
          printf("\n%d\n",tvf[i]);
          for (j=0;j < components;j++)
           {
            fbuf = f_arr[i][j];
            printf("%f ",fbuf);
           }
          printf("\n");
         }
}


/* function cellfunc()                                                     */
/*                                                                         */
/* function initializes data arrays from cell files                        */
/*                                                                         */
/* Input data (data in ascii files cella0.dat through cella5.dat):         */
```

```
/*  f = cell status vector -- TRUE => cell is active                    */
/*  a0 = excitation positioner coordinate vectors                       */
/*  a1 = receiving positioner coordinate vectors                        */
/*  g = number of components in this cell's mixture                     */
/*  string_array = composition strings                                  */
/*  scans = number of scans used for current cell                       */
/*  scount = number of points in spectrum of current cell               */
/*  norm = normalization indicator for current cell                     */
/*  spect = spectrum wave numbers for current cell                      */
/*                                                                      */
/* Output data (parameter arrays sent to calling function):             */
/*  f = cell status vector -- TRUE => cell is active                    */
/*  a0 = excitation positioner coordinate vectors                       */
/*  a1 = receiving positioner coordinate vectors                        */
/*  g = number of components in this cell's mixture                     */
/*  string_array = composition strings                                  */
/*  scans = number of scans used for current cell                       */
/*  scount = number of points in spectrum of current cell               */
/*  norm = normalization indicator for current cell                     */
/*  spect = spectrum wave numbers for current cell                      */
/*                                                                      */
/* Functions called: none                                               */
/*                                                                      */
/************************************************************************/

cellfunc(f,a0,a1,g,string_array,scans,scount,norm,spect)
int f[];/*TRUE => cell is active*/
long a0[][3];/*excite positioner coordinate vectors*/
long a1[][3];/*receive positioner coordinate vectors*/
int g[];/*number of components in this cell's mixture*/
char string_array[][MAXCOMPS][MAXSTR];/*composition strings*/
int scans[];/*number of scans used for current cell*/
int scount[];/*number of points in spectrum of current cell*/
float norm[];/*normalization indicator for current cell*/
int spect[][MAXSPEC];/*spectrum wave numbers for current cell*/

{
        /*local declarations and initializations*/
        char *c;/*string input checker*/
        char s[MAXSTR];/*string input buffer*/
        char check[MAXSTR];/*checkout string buffer*/
        int cellok = TRUE;/*TRUE => cell files opened and had good data*/
        int i;/*loop counter*/
        int d;/*conversion counter*/
        int max;/*place holder*/
        int j;/*current cell number*/
        static char file_list[5][13] = {
                "cella0.dat",
                "cella1.dat",
                "cella2.dat",
                "cella3.dat",
                "cella4.dat"
                        }; /*files containing cell data*/
        char cell[13];/*name of current cell file*/
        FILE *fp,*fopen();/*file handling pointers*/

    /*the function starts doing work here*/
    for (j=0;((j<5) && cellok);j++)
    /*open the jth file in turn*/
    /*copy its data to the jth position in each array*
    /*close each file before opening the next*/
    /*loop does not procede beyond first bad cell*/
     {
        strcpy(cell,(&file_list[0][0]) + (13*j));
        fp = fopen(cell,"r");
```

```
        if (fp != NULL) /*if file opened*/
         {
          d = fscanf(fp,"%d",&f[j]);
          if (d == 1) /*if first character was successfully read*/
           {
            if (f[j] == 1) /*if jth cell is active*/
             {
              d = fscanf(fp,"%ld%ld%ld%ld%ld%ld%d%f%d",&a0[j][0],&a0[j][1],
&a0[j][2],&a1[j][0],&a1[j][1],&a1[j][2],&scans[j],&norm[j],&scount[j]);
              if (d == 9) /*if next 9 data points were successfully read*/
               {
                for (i=0;i<3;i++)
                 printf("\na0[%d][%d]=%ld a1[%d][%d]=%ld\n",
                  j,i,a0[j][i],j,i,a1[j][i]);
                printf("\nNumber of scans is %d\n",scans[j]);
                printf("\nNormalizing marker is %f\n",norm[j]);
                printf("\nNumber of spectrum points is %d\n",scount[j]);
                if (scount[j] > MAXSPEC)
                 {
                  max = MAXSPEC;
                  printf("\nSpectrum points for cell %d > %d\n",j,max);
                  cellok = FALSE;
                 }
                else
                 for (i=0;((i<scount[j]) && cellok);i++)
                  {
                   d = fscanf(fp,"%d",&spect[j][i]);
                   if (d != 1)
                    {
                     cellok = FALSE;
                     printf("\nSpectrum not right for cell %d\n",j);
                    }
                  }
                if (cellok)
                 for (i=0;i<scount[j];i++)
                  printf("\nCell %d spectrum point %d is %d\n",j,i,spect[j][i]);
                if (cellok)
                 {
                  d = fscanf(fp,"%d",&g[j]);
                  if (d != 1)
                   {
                    printf("\nComponent count not correctly read\n");
                    cellok = FALSE;
                   }
                 }
                else
                 printf("\nNo attempt to read component count\n");
                if (cellok)
                 {
                  printf("\nNumber of strings is %d\n",g[j]);
                  if (g[j] > MAXCOMPS) /*if there are more than MAXCOMPS components*/
                   {
                    cellok = FALSE;
                    max = MAXCOMPS;
                    printf("\nNumber of components in cell %d > %d\n",j,max);
                   }
                 }
                if (cellok)
                 {
                  fgets(s,MAXSTR-1,fp); /*read rest of line and ignore*/
                  for (i=0;((i<g[j]) && cellok);i++)
                   {
                    c = fgets(s,MAXSTR-1,fp); /*read next composition line*/
                    if ((c == NULL) || (strlen(s) == 0))
                     {
```

```
                    cellok = FALSE;
                    printf("\nString not right in cell %d\n",j);
                   }
                  else
                   {
                    printf("\nString number %d is %s\n",i,s);
                    strcpy((&string_array[j][0][0]) + (MAXSTR*i),s);
                   }
                 }
               }
              if (cellok)
               for (i=0;i<g[j];i++)
                {
                 strcpy(check,(&string_array[j][0][0]) + (MAXSTR*i));
                 printf("\nstring i=%d j=%d is %s \n",i,j,check);
                }
             }
            else
             {
              printf("\nfscanf cell %d did not make 9 conversions\n",j);
              cellok = FALSE;
             }
           }
          else
           printf("\ncell %d not active\n",j);
         }
        else
         {
          printf("\nfscanf did not make first conversion in cell %d \n",j);
          cellok = FALSE;
         }
        fclose(fp); /*done with jth file -- close it*/
       }
      else
       {
        printf("\nfile %s failed to open\n",cell);
        cellok = FALSE;
       }
    }
    if (cellok)
     printf("\nInput of data from cell files to arrays went well\n");
    else
     printf("\nInput of data from cell files to arrays went badly\n");
    return(cellok);
}


/* function feedback1()                                              */
/*                                                                   */
/* function displays data arrays from cell files                     */
/*                                                                   */
/* Input data (passed in as parameters):                             */
/*  f = cell status vector -- TRUE => cell is active                 */
/*  a0 = excitation positioner coordinate vectors                    */
/*  a1 = receiving positioner coordinate vectors                     */
/*  g = number of components in this cell's mixture                  */
/*  string_array = composition strings                               */
/*  scans = number of scans used for current cell                    */
/*  scount = number of points in spectrum of current cell            */
/*  norm = normalization indicator for current cell                  */
/*  spect = spectrum wave numbers for current cell                   */
/*                                                                   */
/* Output data (displayed to screen):                                */
/*  f = cell status vector -- TRUE => cell is active                 */
/*  a0 = excitation positioner coordinate vectors                    */
```

```
/*  a1 = receiving positioner coordinate vectors                          */
/*  g = number of components in this cell's mixture                       */
/*  string_array = composition strings                                    */
/*  scans = number of scans used for current cell                         */
/*  scount = number of points in spectrum of current cell                 */
/*  norm = normalization indicator for current cell                       */
/*  spect = spectrum wave numbers for current cell                        */
/*                                                                        */
/* Functions called: none                                                 */
/*                                                                        */
/**************************************************************************/


feedback1(f,a0,a1,g,string_array,a,scans,scount,norm,spect)
int f[];/*TRUE => cell is active*/
long a0[][3];/*excite positioner coordinate vectors*/
long a1[][3];/*receive positioner coordinate vectors*/
int g[];/*number of components in this cell's mixture*/
char string_array[][MAXCOMPS][MAXSTR];/*composition strings*/
int a;/*TRUE => input data ok*/
int scans[];/*number of scans used for current cell*/
int scount[];/*number of points in spectrum of current cell*/
float norm[];/*normalization indicator for current cell*/
int spect[][MAXSPEC];/*spectrum wave numbers for current cell*/

{
        int i,j;/*loop counters*/
        char check[MAXSTR];/*check string*/

        if (a)
          {
           for (j=0;j<5;j++)
            {
             printf("\nCell %d : In use flag = %d \n",j,f[j]);
             if (f[j] == 1)
              {
                 for (i=0;i<3;i++)
                  printf("\na0[%d][%d]=%ld a1[%d][%d]=%ld\n",
                   j,i,a0[j][i],j,i,a1[j][i]);
                 printf("\nNumber of scans = %d\n",scans[j]);
                 printf("\nNormalization indicator is %f\n",norm[j]);
                 printf("\n %d points used in spectrum\n",scount[j]);
                 for (i=0;i<scount[j];i++)
                  printf("\nSpectrum point %d is %d\n",i,spect[j][i]);
                 printf("\nNumber of strings is %d\n",g[j]);
                 for (i=0;i<g[j];i++)
                   {
                    strcpy(check,(&string_array[j][0][0]) + (MAXSTR*i));
                    printf("\nstring i=%d j=%d is %s \n",i,j,check);
                   }
              }
            }
           printf("\nInput of data from cell files to arrays went well\n");
          }
        else
         printf("\nInput of data from cell files to arrays went badly\n");
}


/* function set_one()                                                     */
/*                                                                        */
/* function sends strings to the selected positioner to                   */
/*     activate the home limit switches                                   */
/*     move all three axes home                                           */
/*     set default velocities and accelerations                           */
```

```
/*     set to absolute mode                                                    */
/*                                                                             */
/* input data:                                                                 */
/*  address - furnished by calling function                                    */
/*                                                                             */
/* output data:                                                                */
/*  gofrag - passed to calling function                                        */
/*                                                                             */
/* functions called:                                                           */
/*  writecmd() - send a string to the positioner                               */
/*                                                                             */
/*******************************************************************************/

set_one(address,gofrag)
int address;/*I/O address of PC23 board*/
int *gofrag;/*TRUE => all 3 axes found home*/

{
        int i;/*loop counter*/
        static char message[18][8] /*messages to positioner*/
               = {
        "1LDO ", /*activate axis 1 limit switches*/
        "2LDO ", /*activate axis 2 limit switches*/
        "3LDO ", /*activate axis 3 limit switches*/
        "1GH+5 ", /*move axis 1 to home position*/
        "2GH+5 ", /*move axis 2 to home position*/
        "3GH+5 ", /*move axis 3 to home position*/
        "1RC ", /*did axis 1 go home? */
        "2RC ", /*did axis 2 go home? */
        "3RC ", /*did axis 3 go home? */
        "1A10 ", /*set axis 1 acceleration to 10 rev/sec/sec */
        "2A10 ", /*set axis 2 acceleration to 10 rev/sec/sec */
        "3A10 ", /*set axis 3 acceleration to 10 rev/sec/sec */
        "1V10 ", /*set axis 1 velocity to 10 rev/sec */
        "2V10 ", /*set axis 2 velocity to 10 rev/sec */
        "3V10 ", /*set axis 3 velocity to 10 rev/sec */
        "1MPA ", /*set axis 1 to absolute mode */
        "2MPA ", /*set axis 2 to absolute mode */
        "3MPA "  /*set axis 3 to absolute mode */
                };
        char s[81];/*status message from positioner*/
        char motormove[8];/*string sent to positioner*/
        int homeflag = TRUE;/*TRUE => axis went home */

        *gofrag = TRUE;
        for (i=0;i<18;i++)
         {
           strcpy(motormove,(&message[0][0])+(8*i));
           printf("\n");
           printf("String %s sent to positioner\n",motormove);
           writecmd(address,motormove);
           if ((i>5) && (i<9))
              {
             readanswer(address,s);
             printf("\nPositioner response is %s\n",s);
             if ((s[2] == '*') && (s[3] == '@'))
              printf("\nPositioner went home\n");
             else
              {
               printf("\nPositioner did not go home\n");
               *gofrag = FALSE;
              }
            }
         }
}
```

```
/* function initialize()                                                    */
/*                                                                          */
/* Function to reset the PC23 board                                         */
/* This function assures that the board is ready to accept commands from    */
/* program. The PC23 must have its address set for the first value in the   */
/* user data file, or the program will exit. (ie goflag will clear)         */
/*                                                                          */
/* input data:                                                              */
/*  adr - I/O address of port                                               */
/*                                                                          */
/* output data:                                                             */
/*  gofrag - TRUE => port set up ok                                         */
/*                                                                          */
/* functions called: none                                                   */
/*                                                                          */
/****************************************************************************/

initialize(adr,gofrag)
int adr; /*address of PC23 board*/
int *gofrag;/*TRUE => board set up OK */
{
        /* local declarations and initializations*/
        int i=0;/*loop counter*/

        /*function starts here*/

        /* Page 30 step 1 */
        /* STOP mask forces bits 2, 5 and 6 high*/
        /* bit 2 times out and stops the watchdog timer */
        /* bit 5 restarts the watchdog timer */
        /* bit 6 resets the interrupt latch */
        outp(adr+1,STOP); /* stop whatever the port is doing */

        /* Page 30 step 2 */
        /* wait for bit 5 to go high  indicates port fail*/
        while (!((inp(adr+1)) & FAIL))
                ;

        /* Page 30 step 3 */
        /* START mask forces bit 6 high */
        /* this resets the interrupt latch */
        outp(adr+1,START);

        /* Page 30 step 4 */
        /* CB mask forces bits 5 and 6 high */
        /* this resets interrupt latch and watchdog timer */
        outp(adr+1,CB);

        /* Page 30 step 5 */
        /* START_M mask is all bits high but 7 */
        /* RESTART mask is bits 0, 1, 2 and 4  high */
        /* indicates motors not running and input data buffer ready */
        /* loop causes the system to wait for this pattern */
        /* to appear, but it gives up after 32000 iterations */
        while (((inp(adr+1) & START_M) != RESTART) && (i++ != MAXINT))
                ;

        if (i >= MAXINT) /* PC23 timed out */
         {
                printf("Initialization of PC23 timed out\n");
                *gofrag = FALSE;/* will force main() to quit */
         }
        else
         {
```

```
                *gofrag = TRUE; /*all is well - PC23 did initialize*/

                /* Page 30 step 6 */
                /* INTCLR mask is but 5 high */
                /* restarts watchdog timer */
                outp(adr+1,INTCLR);

                /* Page 30 step 7 */
                /* CB mask forces bits 5 and 6 high */
                /* this resets interrupt latch and watchdog timer */
                outp(adr+1,CB);
        }
}

/* writecmd()                                                              */
/*                                                                         */
/* function writes a command string to the PC23 port                       */
/* program writes command to port by sending one character at a time       */
/*                                                                         */
/* input data:                                                             */
/*  adr - address of port                                                  */
/*  msg - command message                                                  */
/*                                                                         */
/* output data:                                                            */
/*  s - individual message characters sent to port                         */
/*                                                                         */
/* functions called:                                                       */
/*  writech() - send one character to port                                 */
/*                                                                         */
/*************************************************************************/

writecmd(adr,msg)
int adr;/*address of PC23 board*/
char msg[];/*command string*/
{
        /*this loop starts by pointing at the initial character in */
        /*the array. it writes the character to the address then advances*/
        /*the string pointer. when the thing pointed at by the string */
        /*pointer is 0, as at the end of string marker, the loop quits */
        while (*msg) /*while thing pointed at is not zero, keep going*/
                writech(adr,*msg++); /*write character being pointed at*/
}

/* function readanswer()                                                   */
/*                                                                         */
/* function reads status request response string from PC23                 */
/*                                                                         */
/* input data:                                                             */
/*  addr - address supplied by calling function                            */
/*  msg - string received from PC23 port                                   */
/*                                                                         */
/* output data:                                                            */
/*  msg - string passed back to calling function                           */
/*                                                                         */
/* functions called:                                                       */
/*  readch - reads one character from port                                 */
/*                                                                         */
/*************************************************************************/

readanswer(adr,msg)
int adr;/* address of PC23 port */
char msg[];/* holder for string */
{
        /*go get a character from the port*/
        /*write it to the address pointed at by msg*/
```

```
        /*advance pointer to next address*/
        /*keep going until you get end of line marker*/
        while ((*msg++ = readch(adr)) != RETURN)
                ;
        *msg = '\0';/*end of string*/
}


/* Third Level - Functions First Called by Second Level Functions       */


/* function writech()                                                   */
/*                                                                      */
/* Function writes a single character to the PC23 port                  */
/*                                                                      */
/* Input data:                                                          */
/*  addr - address of PC23 port                                         */
/*  alpha - character passed in from calling function                   */
/*                                                                      */
/* Output data:                                                         */
/*  alpha - character written to port                                   */
/*                                                                      */
/* Functions called: None                                               */
/*                                                                      */
/************************************************************************/

writech(addr,alpha)
int addr;/*address of PC23 port*/
char alpha;/*character being sent*/
/* Page 30 step 1 - ASCII character passed as parameter */

{
        /* Page 30 step 2 */
        /*wait until bit 4 becomes 1*/
        /* wait for input data buffer to become ready to receive data*/
        while(!(inp(addr+1) & IDB_M))
                ;

        /* Page 30 step 3 */
        outp(addr,alpha);/*send character*/

        /* Page 30 step 4 */
        /*raise bits 4 5 6 */
        /* bit 4 transmit character from buffer to system*/
        /* bit 5 reset watchdog timer */
        /* bit 6 reset interrupt latch */
        outp(addr+1,IDB);/*send command byte*/

        /* Page 30 step 5 */
        /*wait for input data buffer to clear*/
        while(inp(addr+1) & IDB_M)
                ;
        /* Page 30 step 6 */
        /*reset watchdog timer*/
        /*reset interrupt latch*/
        outp(addr+1,CB);

        /* This step not described on page 30 */
        /*wait for input data buffer to become ready to receive data*/
        while(!(inp(addr+1) & IDB_M))
                ;
}

/*function readch()                                                     */
/*                                                                      */
```

```
/* Function reads one character of a PC23 response to a status request    */
/* returns character response.                                            */
/*                                                                        */
/* Input data:                                                            */
/*  addr - address provided by calling function                           */
/*  alpha - character provided by PC23                                    */
/*                                                                        */
/* Output data:                                                           */
/*  alpha - character returned to calling function                        */
/*                                                                        */
/* Functions called: None                                                 */
/*                                                                        */
/**************************************************************************/

readch(addr)
int addr;/*address of PC23 port*/
{
        /* Page 30 step 1 */
        char alpha=0;/*character to be fetched from PC23 and returned*/

        /* Page 30 step 2 */
        /*wait for a 1 at bit 3 */
        /*wait until output data buffer is ready*/
        while(!(inp(addr+1) & ODB))
                ;

        /* Page 30 step 3 */
        alpha = inp(addr);/*fetch a character from output data buffer*/

        /* Page 30 step 4 */
        /* bit 7 I got it */
        /* bit 6 reset interrupt latch */
        /* bit 5 reset watchdog timer */
        outp(addr+1,ACK);/*acknowledge receipt of character*/

        /* Page 30 step 5 */
        /*wait until output data buffer is not ready */
        while(inp(addr+1) & ODB)
                ;

        /* Page 30 step 6 */
        /*reset interrupt latch and watchdog timer*/
        outp(addr+1,CB);
        return(alpha);
}
```

VITA

Stephen W. Kercel was born in Hopewell, Virginia, on September 23, 1947. He was graduated from Thomas Dale High School in Chester, Virginia, in June, 1965. He received a Bachelor of Science degree with Honors in Electrical Engineering from Virginia Polytechnic Institute and State University in June, 1969.

The author was an electrical engineer and systems analyst with The Tennessee Valley Authority until his retirement in August, 1988. During his career, he was the author of several papers on electromagnetic effects in extra high voltage switching stations.

He entered The Graduate School of The University of Tennessee as a full time student in August, 1988. He will receive the Master of Science degree in Electrical Engineering in August, 1990.

Mr. Kercel is a member of Eta Kappa Nu, Tau Beta Pi and Phi Kappa Phi. He is a Senior Member of The Institute of Electrical and Electronic Engineers and a Registered Professional Engineer in The State of Tennessee. He will be employed by The Oak Ridge National Laboratory after graduation.